Verifiable Design of a Satellite-based Train Control System with Petri Nets

Von der Fakultät für Maschinenbau
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung der Würde

eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation

von: M. Sc. Daohua Wu
aus: Guangdong, China

eingereicht am: 08.04.2014
mündliche Prüfung am: 24.07.2014

Gutachter: Prof. Dr.-Ing. Dr. h. c. mult. Eckehard Schnieder
Prof. Dr. Kurt Lautenbach
Assoc. Prof. Dr. Wei Zheng

2014

"Life is like riding a bicycle. To keep your balance, you must keep moving."

Albert Einstein

Acknowledgements

This work would not have been possible without the support from many people. In particular, I would like to express my gratitude to my supervisor Prof. Dr.-Ing. Dr. h. c. mult. Eckehard Schnieder, who provided me with an endless stream of ideas, but equally importantly, encouraged me to take my own direction with this work.

My gratitude also goes to Prof. Dr. Kurt Lautenbach from the University of Koblenz-Landau and Assoc. Prof. Dr. Wei Zheng form Beijing Jiaotong University, who have reviewed this work and provided valuable comments.

Over the past four years I have had the opportunity to discuss my work, especially the parts associated with Petri net theory, with Jörg Rudolf Müller, even after he left the institute. I thank Jörg from the bottom of my heart.

I owe special thanks to Jan Krause from ifak (Institute for Automation and Communication, Magdeburg) for his substantial contribution to the part of test generation based on SPENAT. The collaboration with Jan was a great joy.

I would also like to thank Prof. Dr.-Ing. Karsten Lemmer from the Institute of Transportation Systems of DLR (German Aerospace Center) for his kindness of being the chairman of my oral defence of the dissertation.

I must thank all my colleagues at iVA (Institute for Traffic Safety and Automation Engineering, TU Braunschweig) and all my friends, who have strongly supported me throughout the four-year research in Braunschweig.

Last but definitely not least, I wish to thank my parents Chengqi Wu and Yuandi Zhang, who always stand by me, although they have absolutely no idea what exactly I have been studying.

This work has been financially supported by the CSC (China Scholarship Council) for the first three years and by iVA for the fourth year for which I am grateful.

Braunschweig, August 2014 Daohua Wu

Abstract

Nowadays model-based techniques are widely used in system design and development, especially for safety-critical systems such as train control systems. Given a design model, executable codes could be generated automatically from the model following certain transformation rules. A high-quality model of a system provides a good understanding, a favourable structure, a reasonable scale and abstraction level as well as realistic behaviours with respect to the concurrent operation of independent subsystems. Motivated by this principle, a first Coloured Petri Net (CPN) model of a satellite-based train control system (SatZB) with the capability of continuous simulation is developed employing the BASYSNET method which adopts Petri nets as the means of description during the whole development process.

After establishing the system model, the verification tasks are identified based on the hazard analysis of the train control system. To verify the identified tasks for quality assurance, verification by means of simulation, formal analysis and testing is carried out considering the four representing system properties: function, state, structure and behaviour. For structural analysis, the concept of open nets is proposed to check the reproducibility of empty markings of scenario nets, the existence of dead transitions in the scenario nets, and the terminating states of the scenario nets. The system behaviour, in which states are involved, is investigated by reachability analysis. Unlike the conventional method of reachability analysis by calculating the state space of the Petri net, techniques based on Petri net unfoldings are introduced in this thesis. As to the functional verification, two model-based test generation techniques, i.e., CPN-based and SPENAT (Safe Place Transition Nets with Attributes)-based techniques, are presented.

In this thesis, the proposed methods are exemplified by the application to the on-board module of SatZB model. According to the verification results, no errors were found in the module. Therefore, the confidence in the quality of the on-board module has been significantly increased.

Kurzfassung

Heutzutage werden in zahlreichen Anwendungen modellbasierte Techniken zur Systementwicklung, insbesondere für sicherheitskritische Systeme wie Eisenbahnleit- und -sicherungssysteme, verwendet. Aus einem Design Modell kann dabei ausführbarer Code automatisch nach bestimmten Transformationsregeln generiert werden. Ein hochwertiges Modell des Systems bietet für die Entwicklung ein gutes Verständnis, eine günstige Struktur, eine angemessene Größenordnung und Abstraktionsebene als auch realistische Verhaltensweisen in Bezug auf den gleichzeitigen Betrieb von unabhängigen Subsystemen. Motiviert von dieses Prinzip wird ein erstes Farbige Petri-Netz (CPN)-Modell eines satellitenbasierten Zugsicherungssystem (SatZB) unter Verwendung der BASYSNET Methode entwickelt, der Petri-Netze als Beschreibungsmittel während des gesamten Entwicklungsprozesses nutzt. Dieses Modell bietet die Möglichkeit zur kontinuierlichen Simulation des Systemverhaltens.

Nach der Erstellung des Systemmodells werden die Verifikationsaufgaben auf der Grundlage der Gefährdungsanalyse des Zugsicherungssystems identifiziert. Die abgeleiteten Bedingungen werden zur Qualitätssicherung durch Simulation, formale Analysen und Tests unter Berücksichtigung der vier Systemeigenschaften (Funktion, Zustand, Struktur und Verhalten) verifiziert. Für die Strukturanalyse wird das Konzept der offenen Netzen vorgeschlagen, um die Reproduzierbarkeit der leeren Markierungen der Szenario-Netze, die Existenz der Toten Transitionen in den Szenario-Netze, und die Abschluss Zustände der Szenario-Netze zu prüfen. Das Systemverhalten wird dabei durch Zustände beschrieben und durch eine Erreichbarkeitsanalyse untersucht. Im Gegensatz zu der konventionellen Methode, welche die Erreichbarkeit durch die Berechnung des Zustandsraums des Petri-Netzes analysiert, werden in dieser Arbeit Techniken auf Basis von Petri-Netz-Entfaltung eingeführt. Für die funktionale Verifikation werden zwei modellbasierte Testgenerierungstechniken, eine CPN-basierte und eine SPENAT (Sicheres Petrinetz mit Attributen)basierte, vorgestellt.

In dieser Arbeit werden die vorgeschlagenen Methoden durch die Anwendung auf das On-Board-Modul des SatZB-Modells veranschaulicht. Dabei wurden nach dem Abschluss der Prüfungen keine Fehler im Modul gefunden, wodurch das Vertrauen in die Qualität des On-Board-Moduls deutlich erhöht wurde.

Contents

List of Figures

List of Tables

Abbreviations

BASYSNET	Braunschweig Description, Analysis and SYnthesiS Method based on Petri NETs
BMW	Beschreibungsmittel, Methoden, Werkzeuge
BVA	Boundary Value Analysis
CCS	Calculus of Communicating Systems
CPG	Coloured Petri net Graph
CPNs/CP-nets	Coloured Petri Nets
CPS	Communicating Sequential Processes
CPT	Coloured Petri net Tree
CTCS	China Train Control System
CTL	Computation Tree Logic
ctSPN	continuous time stochastic Petri nets
DB	Deutsche Bahn AG
DSPN	deterministic and stochastic Petri nets
dtDSPN	discrete time deterministic and stochastic Petri nets
dtSPN	discrete time stochastic Petri nets
EP	Equivalence Partitioning
ERA	European Railway Agency
ERTMS	European Rail Traffic Management System
ETCS	European Train Control System
FSM	Finite State Machine
GNSS	Global Navigation Satellite System
GSPN	generalized stochastic Petri nets
HOL	Higher Order Logic
HRG	Hierarchical Reachability Graph

INDUSI/PZB	Induktive Zugsicherung/ Punktförmige Zugbeeinflussung
ISTA	Integration and System Test Automation
ITCS	Incremental Train Control System
ITEA	Information Technology for European Advancement
IUT	Implementation Under Test
iVA	Institute for Traffic Safety and Automation Engineering
Klub-U	Integrated Train Protection System
LOTOS	Language for Temporal Ordering Specification
LPrTPN	Labeled Prioritized Time Petri Nets
LTL	Linear-time Temporal Logic
LZB	Linienförmige Zugbeeinflussung
MA	Movement Authority
MMI	Man Machine Interface
OO	Object-oriented
PN	Petri nets
PNML	Petri Net Markup Language
PROFUND	PROcess FUNctional and Dependability
PrT nets	Predicate Transition nets
P/T nets	Place/Transition nets
RBC	Radio Block Centre
RT	Reachability Tree
RZL	Rechnergestütztes Zugleitsystem
SADT	Structured Analysis and Design Technique
SATLOC	Satellite Based Operation and Management of Local Low Traffics Lines
SatZB	Satellitengestützter Zugleitbetrib
SCC graph	Strongly-Connected-Component graph
SPENAT	Safe Place Transition Nets with Attributes
SPN	stochastic Petri nets
SRS	System Requirements Specification
SST	State Space Tool
SUT	System Under Test
SysML	Systems Modelling Language
TCC	Traffic Control Centre

TPN	timed Petri nets
UIC	Union Internationale des Chemins de Fer
UML	Unified Modelling Language
VDM	Vienna Development Method
VDM-SL	VDM Specification Language

Chapter 1

Introduction

In this chapter, first the state of the art of train control systems and some relevant projects as well as design and development approaches for train control systems are discussed. Second, the challenges of the automated system development, a model-based, tool supported and automated code generation process, are posed. Third, the objectives of this thesis and approaches proposed in this thesis are introduced. Finally, the structure of the thesis is outlined.

1.1 Train Control Systems

Applying the classical model of information chain to a train control system involves two distinct processes between the dispatcher staff/system and the railway vehicles: the transmission of signalling information to the driver in the cab, and the automated influence on the movement of the vehicles in the form of (usually emergency) breaking [1]. Up to 1990, there were more than 20 different train control systems on service across Europe, e.g., LZB (Linienförmige Zugbeeinflussung), INDUSI/PZB (Induktive Zugsicherung/Punktförmige Zugbeeinflussung), TVM [1], [2]. In fact, these systems were developed based on three common ancestors: the French CROCODILE, the American Continuous Cab Signals and the German INDUSI [1]. To standardise the train control systems in a European level, the European Train Control System (ETCS) was promoted by the European Commission, and it is specified for the compliance with the high speed and conventional interoperability directives [3]. After the introduction of high-speed railways in Japan and Europe, China has established its own railway system, the Chinese Train Control System (CTCS) in 2002 [4].

Apart from introducing new train control systems for high-speed railways, a variety of train control systems for low density traffic lines (or secondary lines) have been brought up in

recent years targeting at reducing the considerable cost due to the employment of massive track-side equipments. The ERTMS (European Rail Traffic Management System) Regional project, whose pilot line locates in Borlange, Sweden, is basically a track-side development by means of a centralised control using the GSM-R system to operate the relevant objects (points, level crossings, key locks, shunting areas, etc.) in the infrastructure [5]. Therefore, the ERTMS Regional provides a solution which allows a significant reduction of track-side investments for the secondary lines. According to the cost study (e.g. in the ERTMS Regional project) and the investigation of the GNSS (Global Navigation Satellite System)-based solutions (e.g. in the EU project LOCOPROL [6]) for low density traffic lines, the cost of the on-board and track-side of a train control system for low density traffic lines will alleviate massively when using the GNSS technology. As a consequence, a number of advanced satellite-based train control systems haven been proposed in Europe, such as RZL (Rechnergestütztes Zugleitsystem) [7], [8], SatZB (Satellitengestützter Zugleitbetrib) [9], SATLOC (Satellite Based Operation and Management of Local Low Traffics Lines) [10].

Table 1.1 summaries some train control systems mentioned above.

TABLE 1.1: Train control systems

Line type	High-speed lines		Low density traffic lines (Secondary lines)			
Train control system	ETCS	CTCS	ERTMS Regional	Advanced satellite-based train control system		
				RZL	SatZB	SATLOC
Country/ Region/ Pilot line	Europe	China	Sweden	Austria	Germany	Romania

SatZB. As the system studied in the thesis, the SatZB system is further introduced as follows.

The general architecture of SatZB is shown in Figure 1.1. There are two main components: the on-board subsystem and the traffic control centre (TCC). The TCC manages a list of block sections and gives movement authorities (MA) to the train to drive into defined sections. The on-board subsystem uses GNSS to locate the train and sends location reports to the TCC. When the train approaches the border of its current section, the on-board subsystem sends a request to the TCC for driving into the next section. The communication between the on-board subsystem and the TCC is carried out by mobile radio. Notice that SatZB exploits fixes block sections (see Figure 1.2). Generally, the track between two stations is defined as a block section.

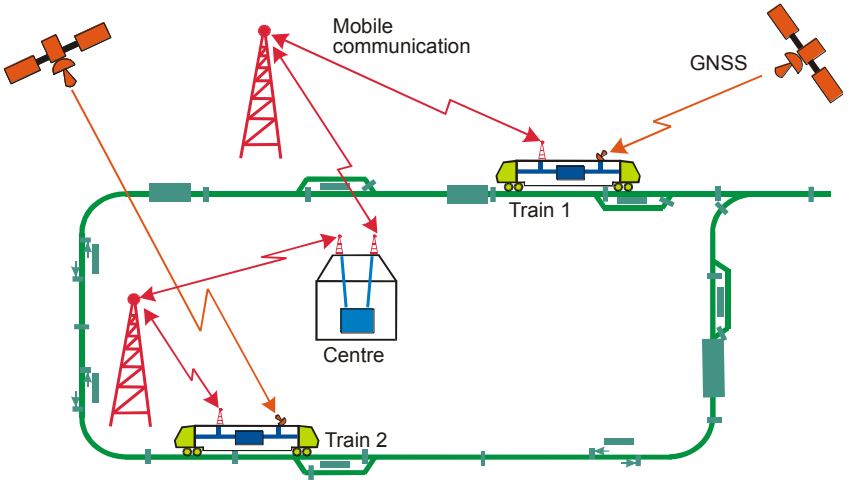


FIGURE 1.1: The general architecture of SatZB [9]

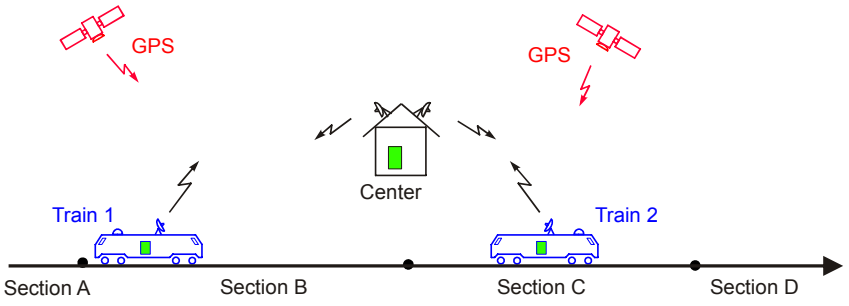


FIGURE 1.2: Fixed block sections

To get a better understanding and a closer investigation of SatZB from both the perspective of software and hardware, the architectures of the on-board subsystem and the TCC presented in [11] are shown in Figure 1.3.

1.2 Design and Development of Train Control Systems

Over the years, numerous methods have been proposed to design and develop various train control systems. In the following, some cases are introduced.

ETCS. Due to the usage of different incompatible train control systems in different countries or companies in Europe, a train that runs through different countries has to be equipped with several systems or the train traction units and drivers have to be changed at each border. The first solution incurs great expense, while the second one is time consuming. To avoid this situation, the ETCS project was issued by the International Union of Railways (Union Internationale des Chemins de Fer, UIC). ETCS is a standard requirement for European train

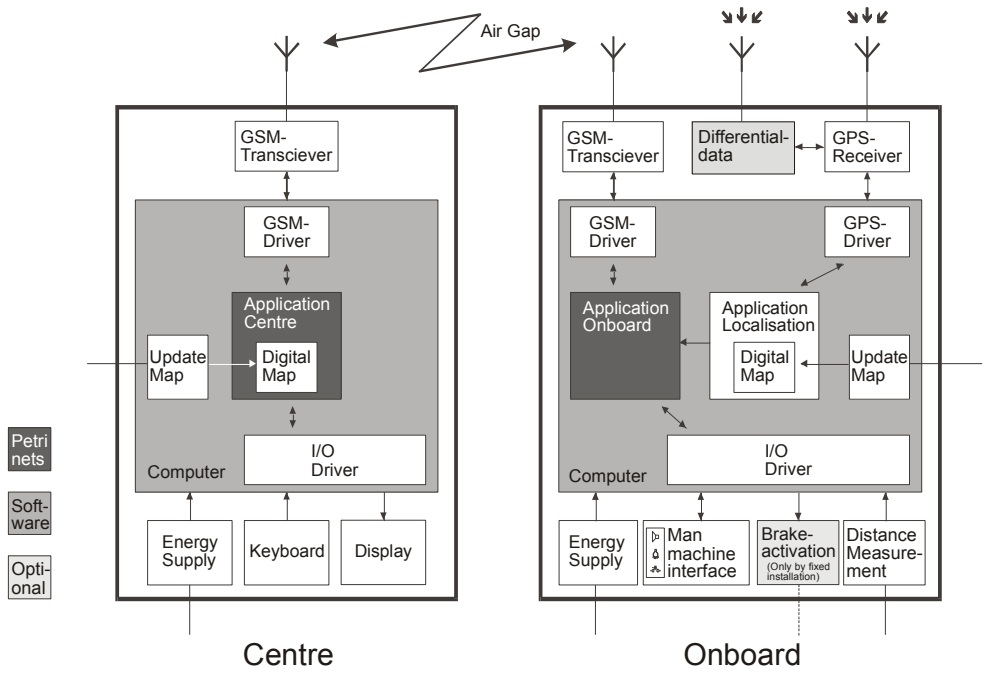


FIGURE 1.3: The architectures of the on-board subsystem and the traffic control centre [11]

control systems and provides a uniform and language-independent signalling information for the Man Machine Interface (MMI). It is mainly dedicated to train interoperability, i.e., it allows an international train running from one country to another without changing either locomotives or drivers [12], [13].

To reflect the architecture of the real system, modelling ETCS in a modularised manner has been introduced in the Institute for Traffic Safety and Automation Engineering (iVA), TU Braunschweig [12], [14]. In [12], the modelling of ETCS starts from three different modelling paradigms: components, scenarios and functions. A train control system model can be divided into different modules (component models) (e.g., the ETCS consists of two main components: the on-board subsystem and the Radio Block Center (RBC)). For structuring the component models of the on-board subsystem and the RBC, the layered approach proposed by [13] is adopted. A component modelled with Coloured Petri Nets (CPNs or CP-nets) [15], in vertical, can be decomposed into scenario nets, function nets and additional nets for the purpose of independent simulation (see Figure 3.5 in chapter 3). In [14] a similar approach is used to model the ETCS. However, the author presented more detailed nets for the context level (system level), process level (component level), scenario level, and functional level with respect to the interfaces of each component and the data exchanged between different components.

openETCS. The release of the ETCS System Requirements Specification 3.0.0 (SRS 3.0.0) (also

called "baseline 3") by the European Railway Agency (ERA) makes the SRS a "public domain" document. It means that everyone in Europe is legally entitled to use the SRS to develop ETCS equipments [16]. With the access to the ETCS SRS, different manufacturers design their own systems according to their own understanding of the textual specification resulting in different versions of ETCS implementations which maybe not fully interoperable. This would make the major goal of unrestricted interoperability of the ETCS be missed. Based on these realities, a currently ongoing project called "openETCS" has been proposed by a group led by DB (Deutsche Bahn AG, Germany) to the ITEA 2 (Information Technology for European Advancement) program in 2012. The purpose of the openETCS project is to develop a framework integrating modelling, development, validation and testing for leveraging the cost-efficient and reliable implementation of ETCS [17]. This framework should be managed by applying the "open proofs" methodology via providing formal system requirements and use case specifications, embedded control software, interfaces, architectures, tools and safety case documents all under an open source license [18]. The framework will provide a holistic tool chain across the whole development process of ETCS software. The tool chain will support the formal specification and verification of the ETCS system requirements, the automatic and ETCS compliant code generation and validation, and the model-based test case generation and execution [17].

Within the application of the methodology of open proofs, the formalisation of the SRS of the on-board subsystem (defined in SUBSET-026 [19]) into a high-level, simulable, semi-formal and formal model using a part of a tool chain is one of the goals of the project [20]. The semi-formal and simulable (but not in real time) model describes all the requirements considered in the scope of system design, covering as many features as possible of the on-board unit. For the chosen parts of the system, the semi-formal models will be completed with strictly formal models, with which complementary elements are provided for the verification and validation using formal proofs. Another goal of the openETCS project is to define the tool chain for developing the on-board software that can fulfil the EN 50128 [21] software development requirements [20]. The executable codes (SIL 4 [22]) of the on-board software are generated from the strictly formal software model. For the tool chain, at present, three proposals have been made by the project: SCADE-based primary tool chain, ERTMS Formal Specs (EFS) [23], [24] based primary tool chain and B [25]. In particular, the SysML/SCADE [26] has been chosen as an already available tool chain for the on-board unit modelling [27].

Satellite-based Train Control Systems. In Europe, a large part of the railway network consists of single track lines most of which are local lines. These lines tend to have relatively low traffic density, and often have obsolete signalling systems or none at all. Consequently, the operational maintenance of these systems becomes more and more difficult and costly and the personnel involvement for normal operations purpose is intensive. This may easily incur

unsatisfactory operation services and even safety issues due to the intensive involvement of personnel. An effective and inexpensive solution for reducing the cost and improving the safety of low traffic density and regional lines is therefore urgently needed. Using GNSS-based localisation for train control is one of such solutions. The already known satellite-based train control systems in operation including Incremental Train Control System (ITCS) in United States of America, Integrated Train Protection System (Klub-U) in Russia and RZL in Austria. The ongoing EU project SATLOC, applied in Romania, is using the GNSS-based localisation technology as well.

The RZL system, developed at the Upper Austrian University of Applied Sciences in Wels, was first equipped in the narrow gauge line from Gmunden to Vorchdorf in the Austrian federal state of Upper Austria in 2003 [8]. In [8], a systematic design based on UML [28] is raised for the development of the RZL system. Based on the system requirements, *use cases* are used for all operational input/output sequences. For each use case, a *sequence diagram* and a textual description with a formalised structure is specified. To design and verify the software tasks that have a cyclic behaviour, *activity diagrams* are adopted. Additionally, *state machines* are used to describe the states of the steps in the activity diagrams. The last step of this UML-based design is the design of the *class diagrams* which document all methods (but not all attributes). However, automatic code generation was not employed for implementing the designed system. For the SATLOC project, the system design is based on the RZL system. Before these projects, another satellite-based train control system, SatZB, has been studied (see [9], [11]) for a period of time. The modelling of SatZB with Petri nets in [11] serves as a case study for the automated system development, whereas in [9] the authors concentrated on distributed simulation for multiple trains.

1.3 Automated System Development–BASYSNET

Conventionally, a train control system is developed step by step as following: in each phase specific means of descriptions and methods are used and the output of each phase is a bunch of textual documents. Due to the possibility that different responsible parties or work teams work for different phases, the responsible parties or work teams for a latter phase have to understand the output from the previous phase first, and take it as the input of the latter phase. Furthermore, transformation from the output of the previous phase to the input of the latter phase is needed if different means of descriptions are used in different phases. This transformation process consumes a lot of effort, and more importantly, could provide errors for the reason of misunderstanding, manual transformation or the utilisation of different techniques.

The conventional developments of ETCS and the developments of the RZL and SATLOC exploit the conventional approaches.

To overcome the inconsistency of the developed system to its requirements specification by conventional approaches, automated system development (e.g., the openETCS project which provides a tool chain to support the system development from system requirements to the executable code generation as well as the verification and validation of the developed system) is promising. The automated system development, in fact, is a model-based system development process, within which executable codes could be generated from the model automatically. Figure 1.4 shows the processes of conventional system development and automated system development. The quality of the system developed with automated system development approach has a huge advantage over the quality of those developed by conventional approaches when formal methods (e.g., methods based on Petri nets, B, VDM [29], [30] or UML/SysML,) are applied to the development process. Formal methods offer a potential to obtain an early integration of verification in the design process as well as to provide more effective verification techniques. Roughly speaking, two branches of formal verification techniques can be distinguished: deductive and model-based methods [31]. With deductive methods, the correctness of the system is determined by properties in a mathematical theory and these properties are proven using tools such as theorem provers and proof checkers. Model-based techniques are based on models describing the system behavior in a mathematical precise and unambiguous manner. The system model with algorithms systematically suggests all states of the system model. This provides the basis for a whole range of verification techniques ranging from an exhaustive exploration (model checking/formal analysis) to experiments with a restrictive set of scenarios in the model (simulation), or in reality (testing).

The BASYSNET (The Braunschweig Description, Analysis and SYnthesiS Method based on Petri NETs) [32], [11], developed at iVA, is a formal method for automated system development. It is a universal method for the design and development of discrete or hybrid discrete-continuous control systems. The BASYSNET method uses Petri nets [33], [15] as a universal means of description during the whole process of different phases, by which the output of the previous phase can be used directly as the input for the succeeding phase. The process is a iterative loop cascade which comprises the entire development process of the specification and implementation of the system, and the model-based quality assurance by means of *simulation, testing* and *formal analysis*. Figure 1.5 shows the process of the BASYSNET method. The squares represent activities which are to be carried out, circles represent inputs and results. Two aspects, specification and implementation of the system and quality assurance for the developed system are taken into consideration in the loop cascade. The upper part of the loop cascade model refers to the aspect of quality assurance. Within each phase, the

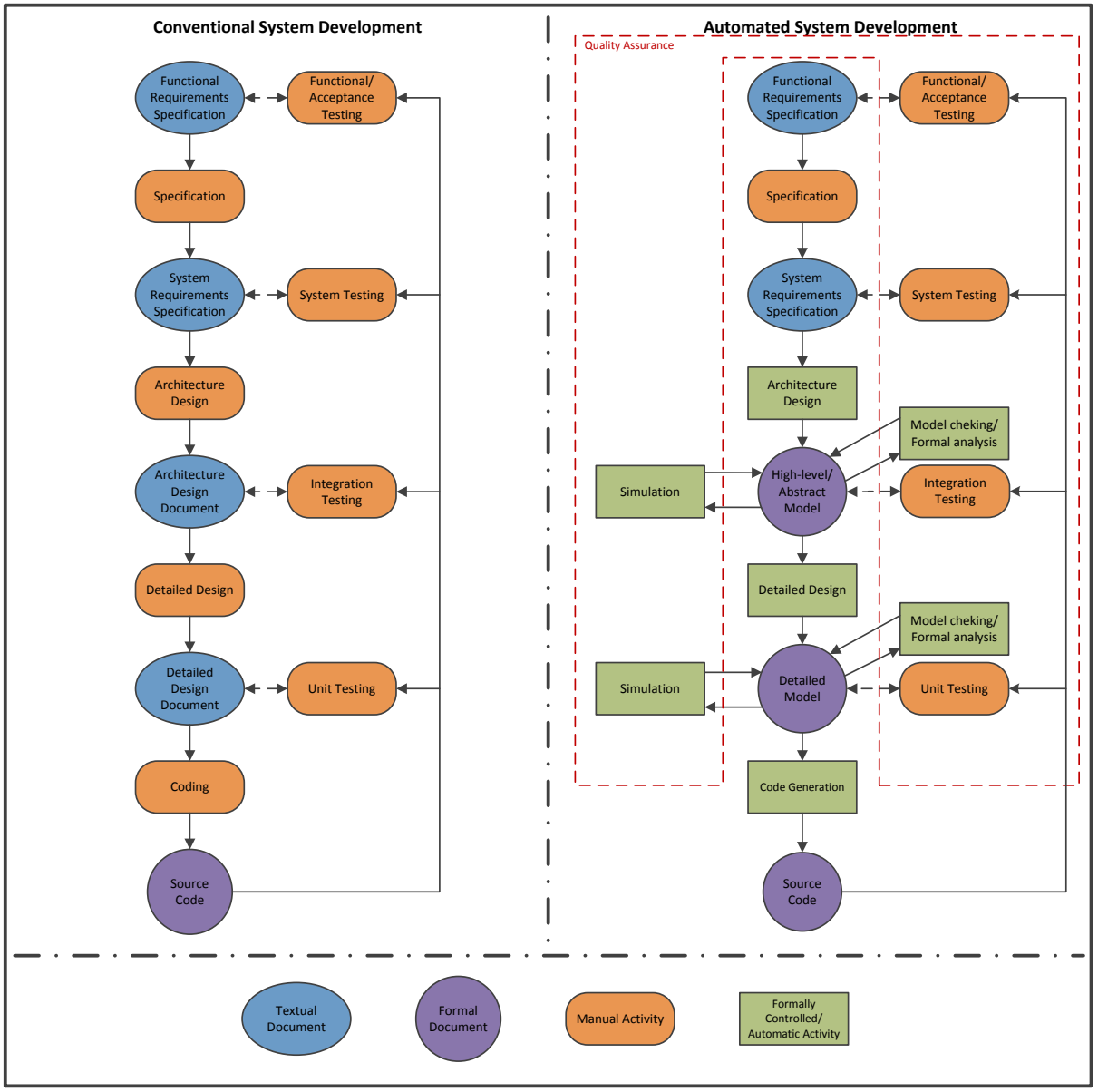


FIGURE 1.4: Conventional system development vs. automated system development

results are checked for the compliance to the results of the previous one. If there exists any difference, returning to an earlier phase is needed.

1.4 Challenges of Verifiable Design of Train Control Systems

Nowadays system development is shifting from using informal textual specification and manual coding techniques to a model-based and tool-supported automated code generation

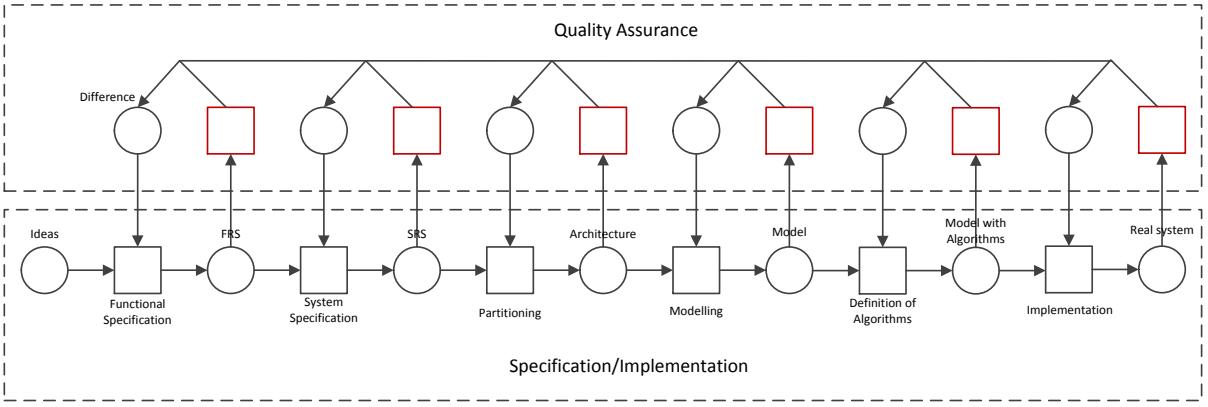


FIGURE 1.5: The BASYSNET method [14]

process. Nevertheless, this automated system development process faces many challenges, especially for the safety-critical systems such as train control systems.

1.4.1 Challenges of the Design

The design of train control systems with formal means of descriptions for the purpose of automatic code generation needs to provide high-quality system models. The quality of a system model in general takes two aspects into account: *completeness* and *the representing system properties* (i.e., *state, function, structure* and *behaviour* [34]). Therefore, to improve and ensure the quality of the train control system model, more information should be included in the model for completeness purpose on one hand, and the verification of the system properties should be carried out on the other hand. Yet the more information is added to the system model, the larger the model will be, and consequently the harder the verification of the system properties becomes. A trade-off between the completeness and the verification complexity of the system model in the phase of system design therefore has to be considered. This is a question of how to properly model a real train control system. In addition, the model is required to have a realistic behaviour of the train control system.

1.4.2 Challenges of the Quality Assurance

According to the BASYSNET method, quality assurance for the design model by verification could be implemented via simulation, testing and formal analysis. For the development of a specific system of train control system, the first challenge of the quality assurance is the *identification of system-specific verification tasks*. To verify the identified tasks, concrete methods

are needed for each manner of the quality assurance. The application of these methods may encounter the challenges of *the state explosion problem, structural and behavioural analysis for large-scale systems, automatic test generation and high test coverage*, etc.

1.5 Objectives and Approaches

Objectives. *How to improve and ensure the correctness and safety of the system?* This is one of the major questions to be answered when one develops a safety-critical system. Answering this question, the following objectives are set in this thesis based on the state-of-the-art developments of train control systems and the presented challenges of model-based and tool-supported automated system development (i.e., the BASYSNET method):

1. *Establish the design model of a satellite-based train control system that is appropriate for system implementation*. Namely, the system model has a reasonable scale as well as a well structure satisfying the readability.

2. *Identify the verification tasks for the design model in consideration of being a safety-critical system*. Hazardous conditions of the design model should be excluded, and functions for safety and normal operations should be verified.

3. *Explore methodologies for carrying out the verification of the system model by means of testing and formal analysis with respect to the identified verification tasks*. For testing, automatic test generation techniques with adequate test coverages are of interest, and for formal analysis, methods of structural analysis within the consideration of large-scale system models and methods of the reachability analysis avoiding the state explosion problem are desired.

Approaches. To cope with the contradiction between the completeness and the verification complexity of a system model, intuitively, two solutions are available. One is to build relatively abstract models by sacrificing the completeness. This kind of abstract models however are not detailed enough for generating executable codes in automated system development. The other solution is to develop a more detailed system model by the manner of modularisation. Different modules or submodules are interconnected by specified interfaces. This is easy to be implemented with CPNs which support the specification of hierarchically structured models. Apparently, the second solution is more suitable for the automated system development. Therefore, a hierarchically modelling approach is employed to model the satellite-based train control system as follows.

- On the top level of the system model, the *object-oriented* approach is used in order to reflect the architecture of the real system. On the second level of the on-board and the TCC module, the *state-oriented* approach is applied based on the concept of scenario. Each scenario of the system, defined from the view of system operators, associates with a scenario nets in the on-board and TCC module. On the third level of the scenario nets, the *process-oriented* approach is exploited to deal with the incoming and outgoing data. On the bottom level, function blocks are specified with the *function-oriented* approach. A function block could be called by different scenario nets. To enable the implementation of the verification by model execution, a synchronisation method is raised using *reference variables* [35].

Given a system model of a train control system, the verification tasks must be identified before conducting the verification activities. Thus, a hazard analysis based approach for the identification of verification tasks is presented.

- On the basis of the *hazard analysis* of the satellite-based train control system, hazardous conditions of the system model, functions of the real system for both safety and normal operation as well as their allocations in the system model are identified, from which the verification tasks of the system model are specified.

To verify the specified tasks of the system model, approaches from both static and dynamic perspectives are proposed.

- For the structural verification, the notion of *open nets* [36], [37] is used to investigate the structural properties of the system model based on the theories of structural properties of Petri nets and reproducibility of empty markings of Petri nets introduced by Lautenbach [38].
- For investigating the behavioural properties of the (hierarchical) CP-nets, the unfolding-based techniques are employed. To unfold a (hierarchical) CP-net directly without transforming into a intermediate low-level Petri net, an extension of [39] is accomplished. The reachability analysis for the CP-nets are discussed on the basis of the concepts of *linearisation* and *configuration* [40] of unfoldings.
- For the functional verification, two model-based test generation techniques based on CPNs and SPENAT (Safe Place Transition Nets with Attributes) [41] are exemplified by generating test cases for the on-board module of the system model. The test model of the CPN-based technique includes a behavioural module of the test object and modules of the environments of the test object. These modules form a *closed system*. The

SPENAT-based technique only model the intended behaviour of the test object, i.e., no environment of the test object has to be modelled because of the possible input/output modelling with a SPENAT. Thereby a SPENAT is an *open system*. By applying two test generation techniques, systematic errors could be avoided, which is vital to safety-critical systems.

1.6 Outline

After this introduction, in the second chapter, Petri nets, both low-level Petri nets including timed Petri nets and CP-nets (high-level Petri nets), are formally introduced as Petri nets are chosen as the means of description for the system design in this thesis. Besides, Petri net tools, e.g., π-Tool [42], Poseidon [43] and CPN-Tools [35], are compared with respect to different features.

In the third chapter, the approach of modelling a satellite-based train control system, in particular, the on-board subsystem with CPNs is elaborated.

In the fourth chapter, the verification tasks of the system model (developed in chapter 3) are specified in consideration of the requirements of a safety-critical system.

In the fifth chapter, Petri net analysing techniques that are used for quality assurance are discussed. These techniques include behavioural analysis, structural analysis and coverage-based test generation.

In the sixth chapter, the structural properties of consistency and controllability (introduced in chapter 5) are verified using the notion of open nets and it is applied to the scenario nets of the on-board subsystem model (developed in chapter 3) according to the verification tasks (specified in chapter 4).

In the seventh chapter, reachability analysis based on Petri net unfoldings is investigated to verify the dynamic behaviour of the system model (introduced in chapter 5) and an application example (derived from chapter 3) is given according to the verification tasks (specified in chapter 4).

In the eighth chapter, two model-based test generation techniques are proposed for testing the functionality (specified in chapter 4) of the on-board subsystem model (developed in chapter 3). The first technique uses CP-nets to establish the test model and generates test cases with the "all state sequences" criterion (introduced in chapter 5). The second technique adopts SPENAT to develop the test model and derives test cases from its unfolding also with the "all path" criterion.

In the last chapter, the conclusions and outlook are provided.

In summary, the process of this thesis starts with two parallel aspects (i.e., theory and application) as depicted in Figure 1.6. The first one is the theory of automated/model-based system development. The second one is the explicit system design of a train control system. At last, the theory is applied to the design of the satellite-based train control system.

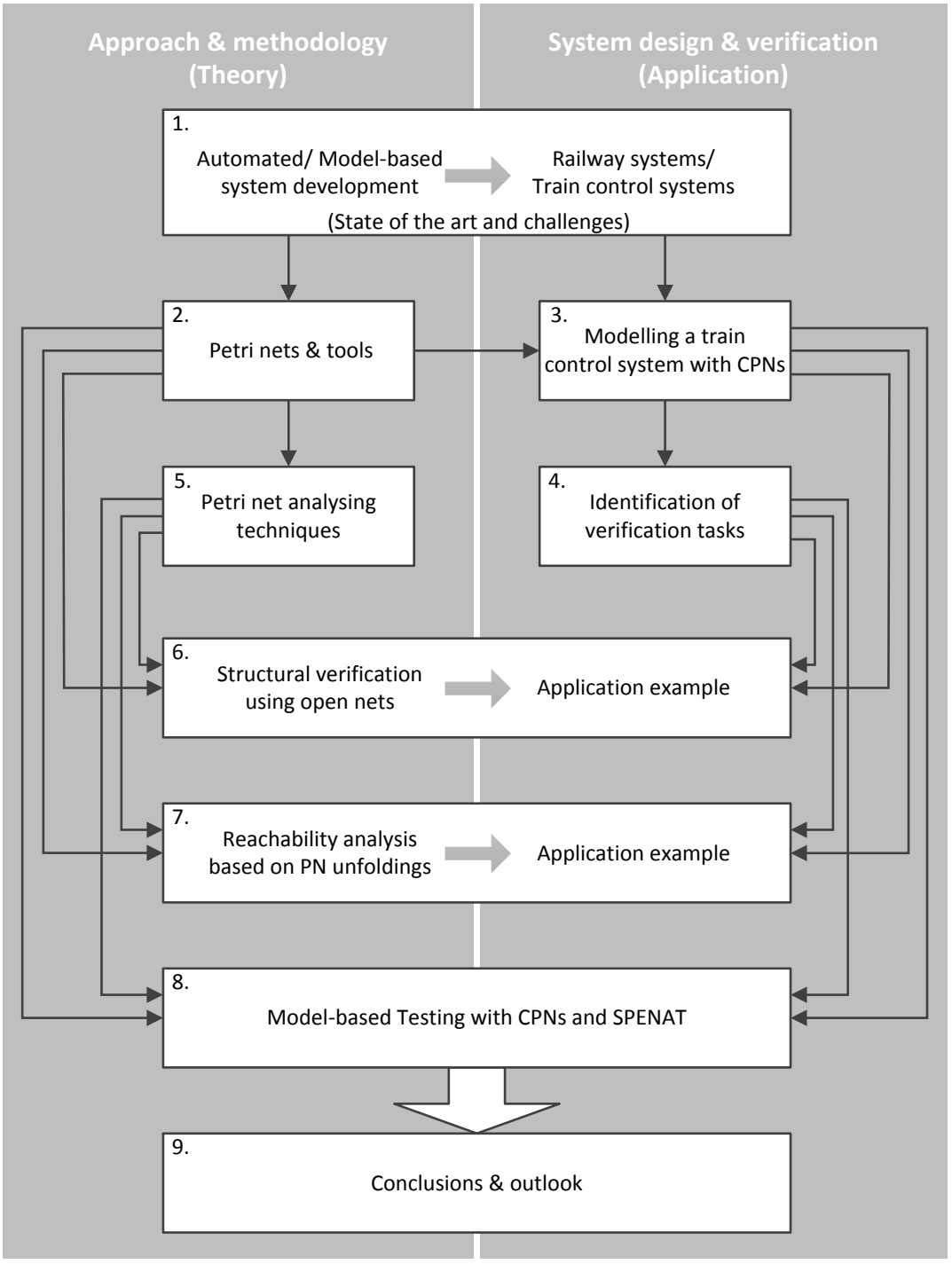


FIGURE 1.6: The process and structure of the thesis

Chapter 2

Petri Nets and Tools

The techniques of *formal methods* are suggested in EN 50128 [21] for software requirements specification, software design and implementation as well as modelling in the area of railway applications. Formal methods provide a means of developing a description of a system at some stage in the phases of requirements specification, design or coding. The resulting description takes a mathematical form and can be subjected to mathematical analysis to detect various classes of inconsistency or incorrectness. A formal method will generally offer a notation (formal language), a technique for deriving a description in that notation, and various forms of analysis for checking a description for different correctness properties. A modelling language is considered as formal if its syntax and semantics are expressed mathematically [44]. Petri nets, named after Carl A. Petri who created in 1962 a net-like mathematical means of description for the study of communication with automata [45], is one of such formal languages.

In this chapter, the formal definitions of Petri nets and Coloured Petri Nets, an overview of timed Petri nets and a survey of some existing Petri net tools are introduced.

2.1 Petri Nets

Petri nets are *graphical* and *mathematical* means of descriptions that provide a uniform environment for modelling, formal analysis and design of discrete-event systems [46], [45]. Petri nets are well-known in describing the systems that characterised as being *concurrent, asynchronous* and *distributed*. As graphical means of descriptions, Petri nets support graphical representations of the system model, which makes it better to understand the requirements specification of the system comparing with the using of textual descriptions. As mathematical

means of descriptions, it is possible to describe a Petri net model by a set of linear algebraic equations, or other mathematical models reflecting the behaviour of the system. This allows one to check some properties of the underlying system by formal analysis independent of simulation which can only show the presence of errors in the model instead of asserting the absence of errors. The ability of Petri nets to verify the model formally is especially important for safety-critical systems such as railway application systems.

Petri nets have been successfully applied on the field of modelling and analysis of communication protocols [47], [48], [49], [50], [15], distributed systems [51], [52], [53], [54], concurrent and parallel programs [49], [55], [56], [57], [58], [59], and other areas.

In this section, Petri nets are formally introduced based on [33], [38], [60], [61], [62], [46].

2.1.1 Basic Definitions

A *(Petri) net* [33] is a triple $N = (P, T, F)$, where P and T are set of *places* and *transitions*, respectively, and $P \cap T = \emptyset$; F is a flow relation $F \subseteq (P \times T) \cup (T \times P)$ for the set of *arcs*. Places and transitions are generally called *nodes*. The *preset* of a node x, denoted by ${}^\bullet x$ is the set $\{y \in P \cup T | (y, x) \in F\}$. The *postset* of a node x, denoted by $x^\bullet$ is the set $\{y \in P \cup T | (x, y) \in F\}$. It is possible to extend this definition to hold for a set $H \subseteq P \cup T$ by ${}^\bullet H := \{y | \exists x \in H, (y, x) \in F\}$ and $H^\bullet := \{y | \exists x \in H, (x, y) \in F\}$.

A *Place/Transition net (P/T net)* [33], [38], [60], [61] is a quadruple $N = (P, T, F, W)$, where (P, T, F) is a net and W is a function $F \to \mathbb{N} \setminus \{0\}$, where $\mathbb{N}$ denotes the natural numbers including 0. If $W : F \to \{1\}$, we write in short $N = (P, T, F)$. In this case, we call the net *ordinary*. A *marking* of a P/T net N is a mapping $P \to \mathbb{N}$. We identify a marking M with the multiset containing $M(p)$ copies of p for every $p \in P$, where $M(p)$ indicates the number of tokens on p under M. For instance, if $P = \{p_1, p_2\}$ and $M(p_1) = 1$, $M(p_2) = 2$, we write $M = \{p_1, p_2, p_2\}$. A transition $t \in T$ is *enabled* under M, if $\forall p \in^\bullet t : M(p) \geqq W(p, t)$. If t is enabled under M, then the transition may fire, thus transform the marking M into a follower marking M' represented by $M[t\rangle M'$, where

$$M'(p) := \begin{cases} M(p), & p \notin^\bullet t \cup t^\bullet \\ M(p) - W(p,t), & p \in^\bullet t \setminus t^\bullet \\ M(p) + W(t,p), & p \in t^\bullet \setminus^\bullet t \\ M(p) - W(p,t) + W(t,p), & p \in^\bullet t \cap t^\bullet. \end{cases}$$

The set of *all reachable markings* $[M_0\rangle$ from a marking M_0 is defined by $M_0 \in [M_0\rangle$ and $M \in [M_0\rangle \wedge M[t\rangle M' \implies M' \in [M_0\rangle$. For $t_1, t_2, \cdots, t_n \in T$, $\sigma = t_1 t_2 \cdots t_n$ is a *firing or occurrence sequence* $\iff \exists M_0, M_1, \cdots, M_n : M_0[t_1\rangle M_1[t_2\rangle \cdots [t_n\rangle M_n$. We denote the firing frequency of

$t \in T$ in σ as $\overline{\sigma}(t)$, the vector $\overline{\sigma}$ of all firing frequencies is called *Parikh vector*. The *incidence matrix* $[N(p,t)]$ of a P/T net is defined by

$$N(p,t) := \begin{cases} W(t,p) - W(p,t), & (p,t) \in F \wedge (t,p) \in F \\ -W(p,t), & (p,t) \in F \wedge (t,p) \notin F \\ W(t,p), & (p,t) \notin F \wedge (t,p) \in F \\ 0, & others. \end{cases}$$

Given a P/T net $N = (P, T, F, W)$ and a marking M_0 of N, we call the pair (N, M_0) as a *Place/Transiton system (P/T system)* or *marked P/T net*, M_0 as the *initial marking*. The system (N, M_0) can be presented as a matrix $[[N]M_0]$.

2.1.2 Invariants and Net Representation

Let $[N]$ be the incidence matrix of a P/T net $N = (P, T, F, W)$. We denote each place vector $i \neq \mathbf{0} \in \mathbb{N}^{|P|}$, which is a solution of the linear equation $i^T \cdot [N] = \mathbf{0}^T$ where $\mathbf{0} \in \mathbb{N}^{|T|}$ as an *S-invariant* of N. We denote each transition vector $j \neq \mathbf{0} \in \mathbb{N}^{|T|}$, which is a solution of the linear equation $[N] \cdot j = \mathbf{0}$ where $\mathbf{0} \in \mathbb{N}^{|P|}$ as a *T-invariant* of N. Note that $|P|$ represents the number of places in the set P.

An S-invariant is a vector i , whose $j\ th$ entry represents a weight $i(j)$ associated with the $j\ th$ place, $j = 1, 2, \cdots, m$, such that the weighted sum of tokens remains the same for all the markings reachable from an initial marking. A T-invariant (if it is realizable) is a vector $\overline{\sigma}$ for a firing sequence σ which leads the marking from M_0 to M_0. In other words, given a T-invariant $\overline{\sigma}$, there exist a marking M_0 and a firing sequence σ leading from M_0 back to M_0. In addition, the $i\ th$ entry of $\overline{\sigma}$ is the number of times that the $i\ th$ transition fires in the firing sequence $\sigma, i = 1, 2, \cdots, n$.

Let j be a transition vector and i a place vector of the net N. We define $\|j\| = \{t \in T | j(t) \neq 0\}$ and $\|i\| = \{p \in P | i(p) \neq 0\}$ the *support* of j and i, respectively. We say that N is covered by j and i respectively, if $\|j\| = T$ and $\|i\| = P$.

Let i be an S-invariant and j a T-invariant of a P/T net $N = (P, T, F, W)$. We define i is *non-negative* if $\forall p \in P : i(p) \geq 0$; i is *positive* if $\forall p \in P : i(p) > 0$; i is *canonical* if the greatest common divisor of the components is 1; i is *minimal (elementary)* if i is canonical and has minimal support. j is *non-negative* if $\forall t \in T : j(t) \geq 0$; j is *positive* if $\forall t \in T : j(t) > 0$; j is *canonical* if the greatest common divisor of the components is 1; j is *minimal (elementary)* if j is canonical and has minimal support.

Let $N = (P, T, F, W)$ be a P/T net, i be an S-invariant of N and j be a T-invariant of N. The *net representation* $N_i = (P_i, T_i, F_i, W_i)$ *of an S-invariant* i is defined by $P_i := \|i\|$; $T_i :=^\bullet P_i \cup P_i^\bullet$; $F_i := F \cap ((P_i \times T_i) \cup (T_i \times P_i))$; W_i is the restriction of W to F_i. The *net representation* $N_j = (P_j, T_j, F_j, W_j)$ *of a T-invariant* j is defined by $T_j := \|j\|$; $P_j :=^\bullet T_j \cup T_j^\bullet$; $F_j := F \cap ((P_j \times T_j) \cup (T_j \times P_j))$; W_j is the restriction of W to F_j.

2.1.3 Traps and Co-traps

Let $N = (P, T, F, W)$ be a P/T net and $H \subseteq P$, then H is a *co-trap* iff ${}^\bullet H \subseteq H^\bullet$; H is a *trap* iff $H^\bullet \subseteq^\bullet H$. A trap (co-trap) H is *minimal* iff no trap (co-trap) H', with $H' \subsetneqq H$ exists. In literature, co-trap is often called *deadlock*, also the term *siphon* is used. Once a trap is marked, it remains marked, and once a co-trap is unmarked, it remains unmarked. The supports $\|i\|$ of all non-negative S-invariants i are traps and co-traps.

2.2 Timed Petri Nets

Adding timing constraints into the Petri net models has gained a wide acceptance in the in the area of *performance and reliability evaluation*. These Petri nets are known as *timed Petri nets* (TPN) [63]. In [64] the firing time are assigned to the transitions of the Petri nets, whereas the time are assigned to the places of the Petri nets in [65].

In this section, we briefly introduce some timed Petri nets that associate firing time with the transitions in the net. Different types of timed Petri nets can be classified depending on the type of firing time. The timed Petri nets in which the firing time delays of the transitions are associated with probability (exponential) distributions are called *stochastic Petri nets* (SPN) [66], [46]. Molloy [47], [67] defined stochastic Petri nets by assigning an exponentially distributed firing rate to each transition for continuous time systems or a geometrically distributed firing rate to each transition for discrete time systems, resulting in *continuous time stochastic Petri nets* (ctSPN) or *discrete time stochastic Petri nets* (dtSPN) and these stochastic Petri nets are isomorphic to homogeneous Markov processes. A normal TPN model associates a fixed (deterministic) time delay with each transition; a normal SPN model associates an exponentially distributed firing time delay with each transition. To bridge the gap between the normal TPN and normal SPN models, Marsan [68] introduced a class of *generalized stochastic Petri Nets* (GSPN) which are derived from standard Petri nets by partitioning the set of transitions into two subsets comprising timed and immediate transitions. An exponentially distributed random firing time is associated with each timed transition, whereas

immediate transitions fire in zero time. Besides, Holliday [69] presented a GSPN model allowing the firing duration to be an arbitrary real number. In [70], Marsan defined *deterministic and stochastic Petri nets* (DSPN) in which transitions can fire after either a deterministic or a random, exponentially distributed firing delay. In [71], Zijal provided a class of *discrete time deterministic and stochastic Petri nets* (dtDSPN), in which transitions fire either without time (immediate transitions) or after a geometrically distributed time and transitions with a deterministic firing delay are a special case.

TABLE 2.1: Classes of timed Petri nets

		Firing time of transitions in the nets			
		Deterministic (Fixed/constant time)	Exponential distribution	Geometrical distribution	Zero time (Immediate transitions)
Timed Petri nets	normal TPN	x			
	normal SPN		x		
	GSPN		x		x
	ctSPN		x		
	dtSPN			x	
	DSPN	x	x		
	dtDSPN	x		x	x

2.3 Coloured Petri Nets

The classes of modelling formalisms based on Petri nets are differed in the amount of information represented by the markings of places. The Petri nets that places have boolean markings meaning that a place is either empty, or contains a single token with no value, or holds an integer number of tokens that are anonymous, i.e., one token is not distinguishable from another, are often referred to as *low-level Petri nets*, whereas Petri nets that tokens have values of primitive or complex data types, e.g., integers, strings and records, are as *high-level Petri nets* [44]. The previously presented Petri nets and Place/Transition nets in this chapter are low-level Petri nets while Coloured Petri Nets (CP-nets or CPNs) [15] belongs to high-level Petri nets. Other high-level Petri nets include M-nets [72], [73], [74], and predicate/transition nets [46], etc.

CPNs is a discrete-event modelling language combining Petri nets and the functional programming language CPN ML [15] which is based on Standard ML [75]. It applies the modelling concepts of time, hierarchy and inscription language[44].

CP-nets add timing specification that makes it possible to describe the duration of the execution of activities in the system. This is represented by the addition of integer time stamps to individual tokens. Intuitively, the time stamp in a token can be seen as a model time at which the token is available for consumption from a place. The time concept also means that CP-nets can be applied for simulation-based performance analysis, investigating performance measures such as delays, throughput,and queue lengths in the system, and for modelling and validation of real-time systems [49].

CP-nets support the specification of hierarchically structured models, which makes it possible to work with different levels of detail and abstraction. This is in particular useful for large-scale systems. Constructing hierarchically structured models is realised by using *substitution transitions* and *fusion places*[15]. A substitution transition is a special transition that represents an instance of another CPN module. A fusion place is a member of a fusion set which allows places in different modules to be glued together into one compound place across the hierarchical structure of the model. The fusion places that are members of a fusion set represent a single compound place. CP-nets with substitution transitions are nets with multiple layers of detail. Each substitution transition is related to a more detailed page representing a module or submodule. With substitution transitions, one can have a somewhat simplified net that gives a broad overview (e.g., the top level) of the system you are modelling. By substituting transitions in a specific level net with more detailed pages, more and more details can be brought into the model. This is so-called *hierarchical structuring mechanism* [15], which allows a module to have submodules, a set of modules to be composed to form a new module, and reuse of submodules in different parts of the model.

CP-nets use the CPN ML language to specify declarations and net inscriptions. The inscriptions are used for data manipulation, function definition, and so on. The CPN ML language is an extension of the functional programming language of Standard ML.

In the following, the formal definitions of non-hierarchical and hierarchical CP-nets are adopted from [15].

A *non-hierarchical Coloured Petri Net* is a nine-tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, where:

1. P is a finite set of places.
2. T is a finite set of transitions such that $P \cap T = \emptyset$.
3. $A \subseteq P \times T \cup T \times P$ is a set of directed arcs.
4. Σ is a finite set of non-empty colour sets.
5. V is a finite set of typed variables such that $Type[v] \in \Sigma$ for all variables $v \in V$.

6. $C : P \rightarrow \Sigma$ is a colour set function that assigns a colour set to each place.

7. $G : T \rightarrow EXPR_V$ is a guard function that assigns a guard to each transition t such that $Type[G(t)] = Bool$.

8. $E : A \rightarrow EXPR_V$ is an arc expression function that assigns an arc expression to each arc a such that $Type[E(a)] = C(p)_{MS}$[1], where p is the place connected to the arc a.

9. $I : P \rightarrow EXPR_\emptyset$ is an initialisation function that assigns an initialisation expression to each place p such that $Type[I(p)] = C(p)_{MS}$.

A *hierarchical Coloured Petri Net* is a four-tuple $CPN_H = (S, SM, PS, FS)$ where:

1. S is a finite set of *modules*. Each module is a *Coloured Petri Net Module* $s = ((P^s, T^s, A^s, \Sigma^s, V^s, C^s, G^s, E^s, I^s), T^s_{sub}, P^s_{port}, PT^s)$. It is required that $(P^{s_1} \cup T^{s_1}) \cap (P^{s_2} \cup T^{s_2}) = \emptyset$ for all $s_1, s_2 \in S$ such that $s_1 \neq s_2$.

2. $SM : T_{sub} \rightarrow S$is a submodule function that assigns a *submodule* to each substitution transition. It is required that the *module hierarchy* [2] is acyclic.

3. PS is a portsocket relation function that assigns a *portsocket relation* $PS(t) \subseteq P_{sock}(t) \times P^{SM(t)}_{port}$ to each substitution transition t. It is required that $ST(p) = PT(p'), C(p) = C(p')$ and $I(p)\langle\rangle$ for all $(p, p') \in PS(t)$ and all $t \in T_{sub}$.

4. $FS \subseteq 2^P$ is set of non-empty *fusion sets* such that $C(p) = C(p')$ and $I(p)\langle\rangle = I(p')\langle\rangle$ for all $p, p' \in fs$ and all $fs \in FS$.

2.4 Tools

In practical, the use of computer-aided tools is a necessity for the application of Petri nets. There are a variety of Petri net tools in the market and research groups. The "Petri Nets World: Online Services for the International Petri Nets Community" [76], maintained by the TGI group at the University of Hamburg (Germany), has conducted a survey of Petri net tools. Based on this survey and the usage of this thesis, some of the tools are given in Table 2.2. The features of the Petri net tools are defined as follows:

[1] MS refers to "multiset". A *multiset* m over a non-empty set S can be viewed as a function from S into the set of non-negative numbers $\mathbb{N}$. The function maps each element s into the number of appearances, $m(s)$, of the element s in the multiset m [15].

[2] A module hierarchy illustrates the relationship between modules in a hierarchical model can be represented as a directed graph which has a node for each module and an labelled arc for each substitution transition [15].

- **Graphical editor** : the tool has a graphical user interface which supports editing of nets in a graphical representation.
- **Token game animation**: the tool supports simulation with animation of the flow of tokens.
- **Simulation**: the tool supports simulation without graphics to allow maximum simulation performance.
- **State spaces**: the tool supports the generation of state spaces (also known as reachability graphs/trees and occurrence graphs).
- **S-invariants**: the tool supports S-invariants identification.
- **T-invariants**: the tool supports T-invariants identification.
- **Performance analysis**: the tool supports performance analysis such as simulation with time.
- **Unfolding**: the tool supports the generation of unfoldings (or finite complete prefixes).

2.5 Summary

This chapter first introduces Petri nets from the viewpoint of formal methods, and then gives an overview of the advantages of Petri nets as graphical and mathematical means of descriptions in describing the systems characterised as being concurrent, asynchronous and distributed. Literatures referring to the application of Petri nets are presented afterwords. Followed by this, the definitions of Petri nets including Place/Transition nets and Coloured Petri Nets as well as their behavioural properties are formally introduced. Additionally, an overview of timed Petri nets, i.e., various classes of timed Petri nets has been provided. Considering the fact that computer-aided tools are needed for the application of Petri nets, a comparison of some existing Petri net tools has been conducted in the end.

TABLE 2.2: Petri net tools

Tools	Resources	PN supported				Features							
		Low-level Petri nets	High-level Petri nets	Stochastic Petri nets	Petri nets with time	Graphical editor	Token game animation	Simulation	State spaces	S-invariants	T-invariants	Performance analysis	Unfolding
CPN Tools [35]	Aarhus University (DK)		x		x	x	x	x	x			x	
π-Tool [42]	Institute for Quality, Safety and Transportation (iQST) (DE)	x		x	x	x	x	x	x			x	
Poseidon [43]	Universität in Koblenz-Landau (DE)	x				x	x			x	x		
PIPE2 [77]	Imperial College London (UK)	x				x	x	x	x	x	x	x	
TimeNET [78]	Technische Universität llmenau (DE)	x	x	x	x	x	x	x		x		x	
AlPiNA [79]	University of Geneva (CH)		x			x	x	x				x	
CPN-AMI [80]	Université Pierre & Marie Curie (FR)	x	x			x		x	x	x	x		
INA [81]	Humboldt-Universität zu Berlin (DE)	x	x		x				x	x	x	x	
GreatSPN [82]	Università di Torino (IT)		x	x	x	x	x	x	x	x	x	x	
Cunf [83]	César Rodríguez (FR)	x				x							x
VIP Tool [84]	FernUniversität in Hagen (DE)	x				x	x	x					x

Chapter 3

Modelling with Coloured Petri Nets

Modelling is a vital activity in automated system development. The quality of the model has a significant impact on the developed system. A high-quality model provides a good understanding, a favourable structure, a reasonable scale and abstraction as well as realistic behaviours according to the concurrent operation of independent subsystems. For this purpose, a hierarchically modelling approach is employed to model the SatZB system.

In this chapter, first the "BMW" principle is discussed. Second, the approach for realistic modelling of the SatZB system is presented. Last, the CPN model of the on-board subsystem is elaborated for illustrating the proposed approach.

3.1 BMW Principle

In railway domain, a variety of modelling languages, including formal and semi-formal languages have been applied to describe railway application systems, e.g., B in [85], UML in [86], VDM in [87] and Petri nets in [14]. In accordance with each modelling language, appropriate methods and computer-aided tools are employed. For each of theses modelling approaches, based on the so-called "BMW" (in German "Beschreibungsmittel, Methoden, Werkzeuge") principle [32], [88], [34]: *means of description; methods for design and analysis; tools for advanced system engineering to support methods and description*, verification for the design model by different manners can be carried out. These verification manners include formal proof, structural analysis, simulation, testing, etc. *Formal proof* uses theoretical and mathematical models and rules to prove the correctness of a system model or program without executing it; *structural analysis* verifies the structural properties of the system model that are independent of the states of the system; *simulation* imitates the operation of a real-world process or system

over time with the help of the system model; *testing* aims at demonstrating the compliance of the actual and intended behaviours of the system by executing the test suites on the system model. Table 3.1 shows some examples of modelling the railway application systems based on the BMW principle. More formal methods such as CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM and Z are suggested and described in EN50128 [21].

TABLE 3.1: BMW principle applied on railway application systems (VDM: Vienna Development Method; OO: Object-oriented; SADT: Structured Analysis and Design Technique; BASYSNET: Braunschweig Description, Analysis and SYnthesiS Method based on Petri NETs; UML:Unified Modelling Language; VDM-SL: VDM Specification Language.)

BMW			Verification				Code generation	Project/Line/Case study	Source
Means of description	Method	Tools	Formal proof	Structural analysis	Simulation	Testing			
B	The B method	ATELIER B	x				x	French national network (SNCF), Paris suburban network (RER), Underground of Paris (project Météor), Cairo, Calcutta and Lyon	[85], [89], [90]
Diagrams	SADT	IDEF0						GNSS supported low traffic density lines	[91], [92]
UML	OO	Rhapsody			x		x	RZL (Rechnergestütztes Zugleitsystem)	[93], [86], [94], [8], [7]
Coloured Petri nets	BASYSNET	Design/CPN (has been replaced by CPN Tools), Artifex	x	x	x	x	x	SatZB, ETCS	[11], [32], [95], [14], [76], [96]
VDM-SL, VDM++, UML+ VDM++	VDM	VDM Tools	x				x	Interlocking system	[87], [97], [30], [29]

In this chapter, the BASYSNET method (see subchapter 1.3) is adopted to develop the SatZB model in general since Petri nets are used as the means of descriptions during the whole development process. The relationship between the elements of a Petri net system and the four representing properties of its underlying system can be illustrated by the UML class diagram

shown in Figure 3.1. The concept of system can be described by the notation of Petri nets [46], [33]. A Petri net model (P/T system) basically comprises four elements: *place*, *transition*, *arc* and *marking*. The states (marked or not marked by tokens) of the places represent the *state* of the underlying system; a *function* of the underlying system is realised by firing certain transitions in the Petri net model; seeing the Petri net itself as a system, the *structure* of the Petri net model is illustrated by the constructing with places, transitions and arcs; the *behaviour* of the underlying system in the Petri net model relates to the markings of the model and the transitions of markings that are triggered by transition firings.

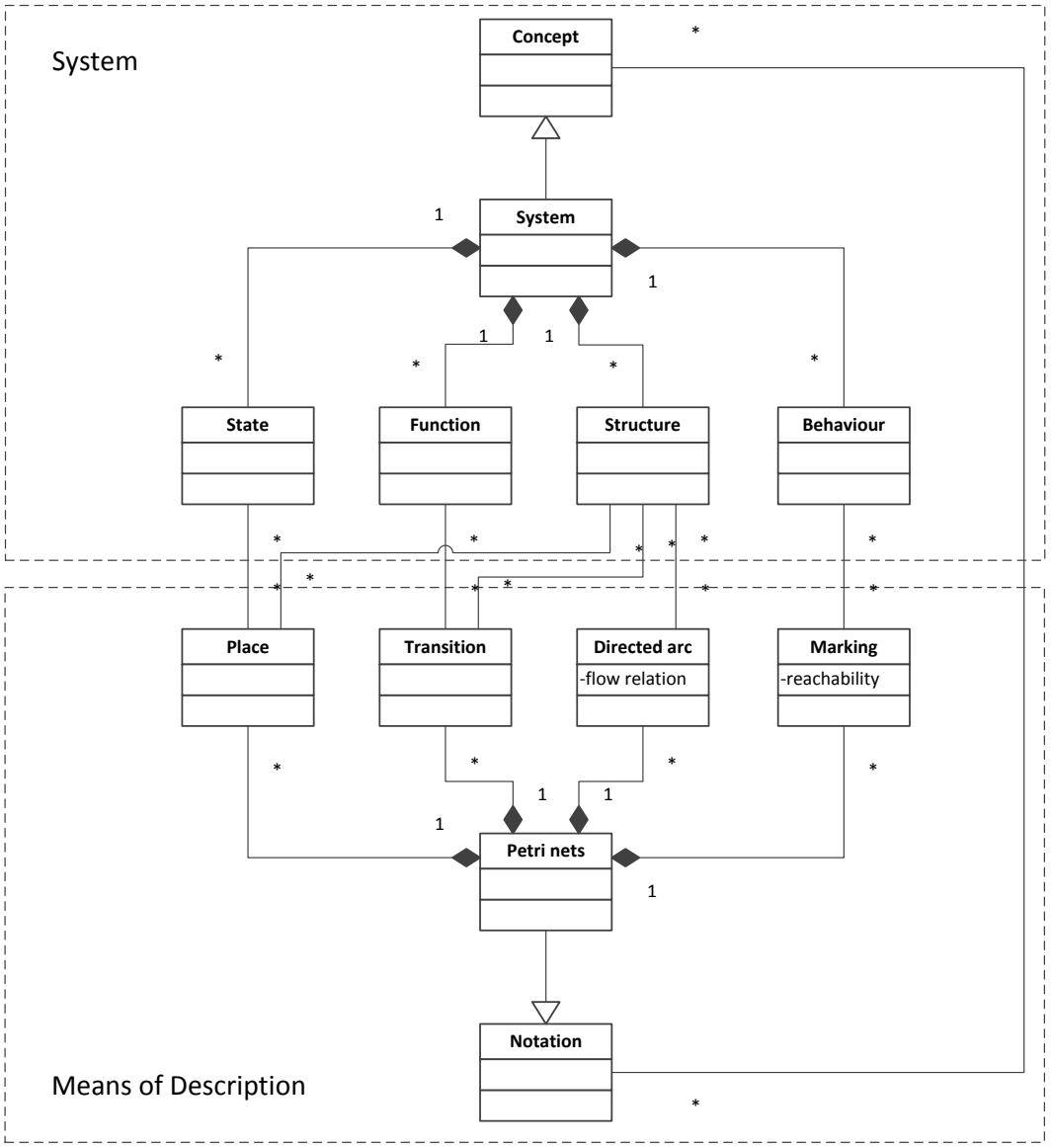


FIGURE 3.1: Petri nets and the underlying systems

With the BASYSNET method, the system model in each phase is executable with the help

of certain Petri net tools such as CPN Tools. *Simulation* by executing the model is appropriate for validation, especially in the early stages of the system model. *Testing* for the system model is also accomplished by model execution, and further, the test suites can be applied to the implementation of the system if specific refinements of the test suites are performed. Verification by means of *formal analysis* is initiated to prove specific properties (e.g., structural and reachability properties) of the system model, which have close relations to the system behaviours. The relationship between the representing system properties of a Petri net model and the methods for system verification is depicted in Figure 3.2.

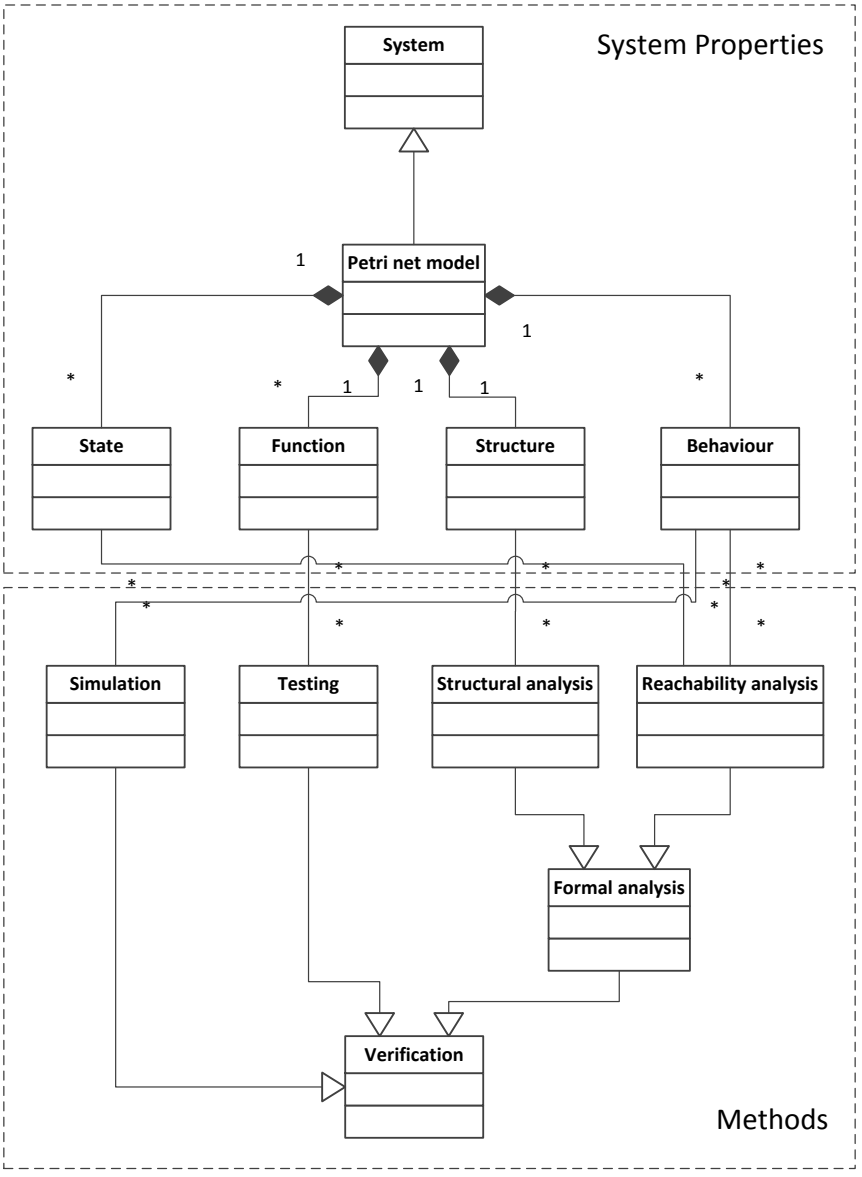


FIGURE 3.2: Methods for system verification with respect to the corresponding properties of a Petri net model

3.2 Modelling Approach

The behaviour of a system can be modelled from different views for different purposes. For the purpose of system design and development on one hand, the system model of a train control system should have the ability to offer the expected operations which consist of a number of scenarios from the view of an operator (customer); on the other hand, a system (model) has four basic representing properties: function, state, behaviour and structure, from the view of an analyst (system engineer). There exists relations between the abstract model that from the view of an analyst and the operation model that from the view of an operator, which is illustrated in Figure 3.3 using the description of UML class diagrams. It is shown that a scenario has a few system functions, comprises a series of system states and is a specific system behaviour indicating by the transitions of system states. A scenario may also relate with other scenarios. As a system itself and being a component of a larger system, a subsystem also has the mentioned four properties.

The definitions of the representing properties of a system, given in [98], are defined as follows.

State. The *state* of a system is the condition of the system at a given instant that determines the potential future sequences of actions that the system may be involved in.

Structure. A system has a *structure*. It consists of a set of parts which have reciprocal relationships to each other as well as to the environment.

Function. A system has a *function*, which can be seen as a specified purpose of the system or its inherent physical state transitions. Following the principle of decomposition, a function can be further divided. Each of those subordinate functions contributes to the overall purpose of the system.

Behaviour. The *behaviour* of a system is a collection of actions that the system may take part in, together with the set of constraints on when those actions can occur. Following the ideas of decomposition and encapsulation, the actions can be either interactions of the system with its environment or internal actions of the system.

3.2.1 Modelling Paradigms

To develop a system model with realistic behaviours of its underlying system, following modelling paradigms are used to model the satellite-based train control system.

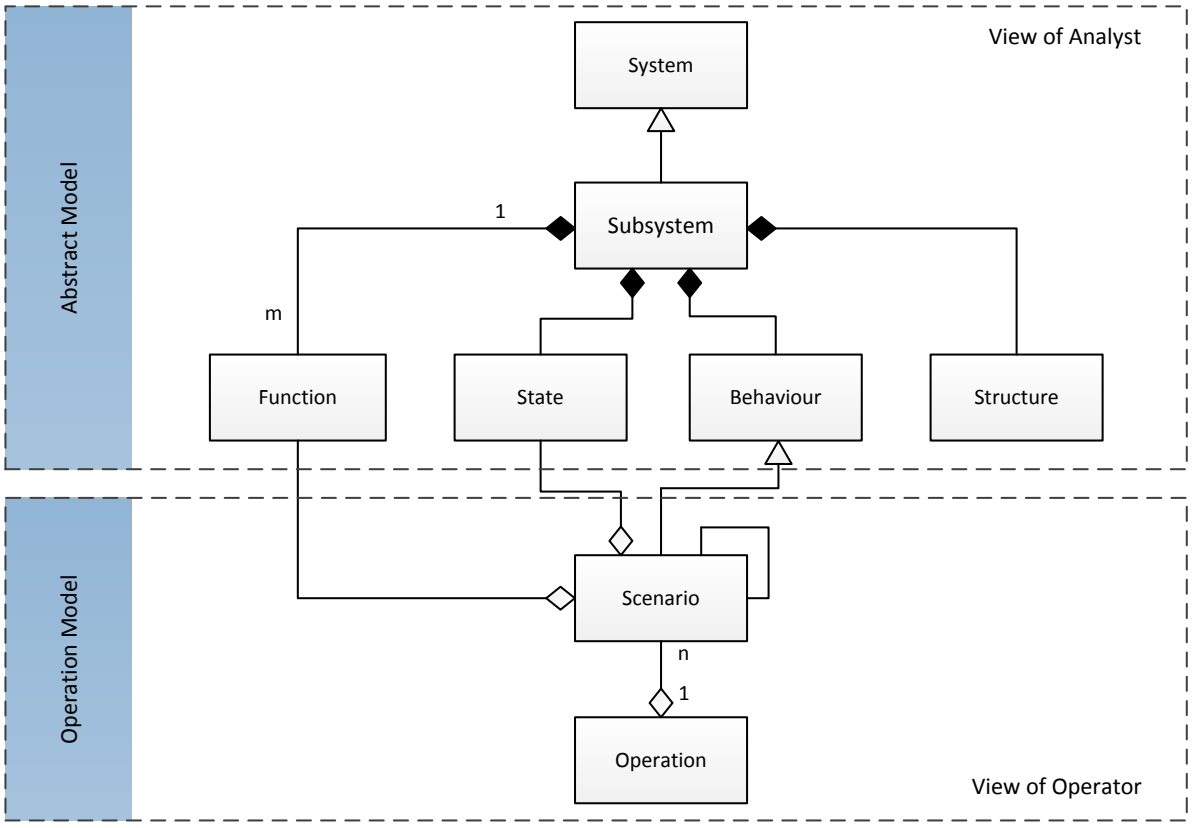


FIGURE 3.3: Diagram of the abstract and operation model of train control systems

Subsystem. The decomposition of a railway system usually turns out to be several different *subsystems*. Generally speaking, a subsystem is a relatively independent component of a system. From the viewpoint of this level, the communication and interaction between the different subsystems are of interest.

Scenario. *Scenarios* define operational situations and processes with respect to the environment of a system. They show the behaviour how the system and its components (subsystems) interact with each other and the environment. A scenario is a description of the behaviour of a system or a collection of components embedded in a comprehensive operational process. It is determined in an unambiguous way by specific starting conditions and a sequence of states and environment events of the system or components in focus [37].

Function. From the perspective of model hierarchy, functions are represented in a lower level in comparison with scenarios. *Functions* are specifically associated with the process aspect of a component. Functional modules can be called by different scenarios and are hence called "functional blocks" [12] which have standard interfaces of input and output.

3.2.2 Model Architecture

In order to reflect the architecture of the real system, the two main subsystems, the on-board subsystem and the traffic control centre are modelled with two modules represented by two *substitution transitions* [1] on the top level of the CPN model. Beside the two main subsystems, the communication system between the on-board subsystem and the traffic control center is modelled with an independent module. Additionally, the localisation unit, an on board unit that localise the train using the GNSS data, has been considered as a subsystem independent of the on-board subsystem in the model, by which it is easier to manage the system model and it is possible to develop the model of the localisation unit and the on-board subsystem separately. Moreover, the module `Train Movement` is developed to imitate the movement of the train so as to generate dummy GNSS data. However, for convenience sake, the connection between the module `On-board Subsystem` and the module `Train Movement` representing by the dot lines has not been established in first model of this work. Instead, we assume that the train moves as time goes on, which is realised by setting a clock inside the `Train Movement` module. The communication channel from the module `On-board Subsystem` to the module `Localisation Unit` only sends a start signal. Figure 3.4 depicts the top level of the CPN model. A substitution transition represents a relatively independent subsystem or component. Places are interfaces of the subsystems or components and the tokens on the places represent the messages or data transmitted between the subsystems or components. The arrows of the arcs indicate the type of the interfaces: input, output or input/output. For simplification purpose, we only use input and output interfaces on the top level of the CPN model.

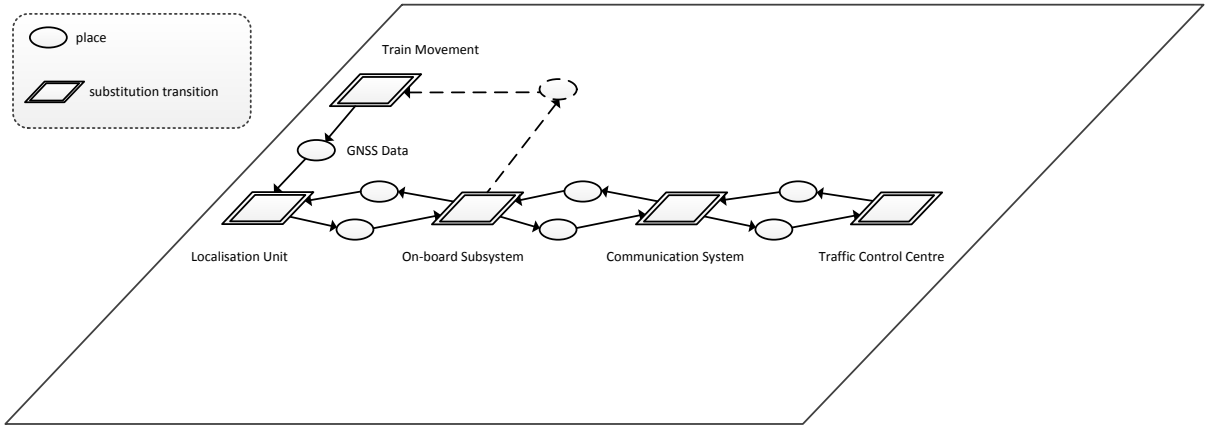


FIGURE 3.4: Architecture of the system model

[1] A substitution transition is a special transition that represents an instance of another CPN module.

3.2.3 Vertical Decomposition of the Main Subsystem Models

In general, the functionalities of the on-board subsystem and the traffic control centre of the satellite-based train control system are similar to the on-board subsystem and Radio Block Centre (RBC) of the ETCS except that the on-board subsystem of the satellite-based train control system uses GNSS data to localise the train and the on-board subsystem of ETCS employs balises instead. Therefore, the approach of establishing a hierarchical model of ETCS with Design/CPN, introduced by Jansen [12] is exploited for vertical decomposition of the main subsystem models (i.e., the on-board subsystem model and the model of the traffic control centre) here.

The decomposition of the main subsystem models is depicted in Figure 3.5. There exists a top level net (net I) for each subsystem model. It is an interface net showing the interfaces of the subsystem. On the second level, an application net A is established for each subsystem, nets P_k and P'_k represent the functions of preprocessing the incoming messages and postprocessing the outgoing messages, respectively. On the third level, the net D is used to distinguish the function decomposition net D_F from the scenario decomposition net D_S, through which the function net F_j is coordinated to the scenarios net S_i on the bottom level. A scenario net S_i can call the function net F_j by applying fusion places, and different scenario nets can call the same function net. Coloured Petri Nets support a method for defining sets of places so that anything that happens to each place in a set also happens to all the other places in the set. The places are then functionally identical. Such places are called *fusion places*, and a set of fusion places is a *fusion set*. These places serve as internal message channels for the calling of function nets and returning results to scenario nets. Nets T_0 and T_k are added for the preliminary simulation purpose. By adding these nets, the subsystem model can simulate separately, which fulfils the needs of the development of distributed systems. If the nets like T_k are substituted by the application nets for other subsystem models, then the net I also shows the connections of this subsystem to other subsystems and the corresponding communication channels on an abstract level.

3.2.4 Switchovers of Scenario Nets

In the functional requirements specification of SatZB, scenarios are usually used to describe a number of operational processes, in which sets of functions are involved. For instance, following text selected from the specification describes the scenario that releases a block section for the train's running. We call this scenario RUNNING.

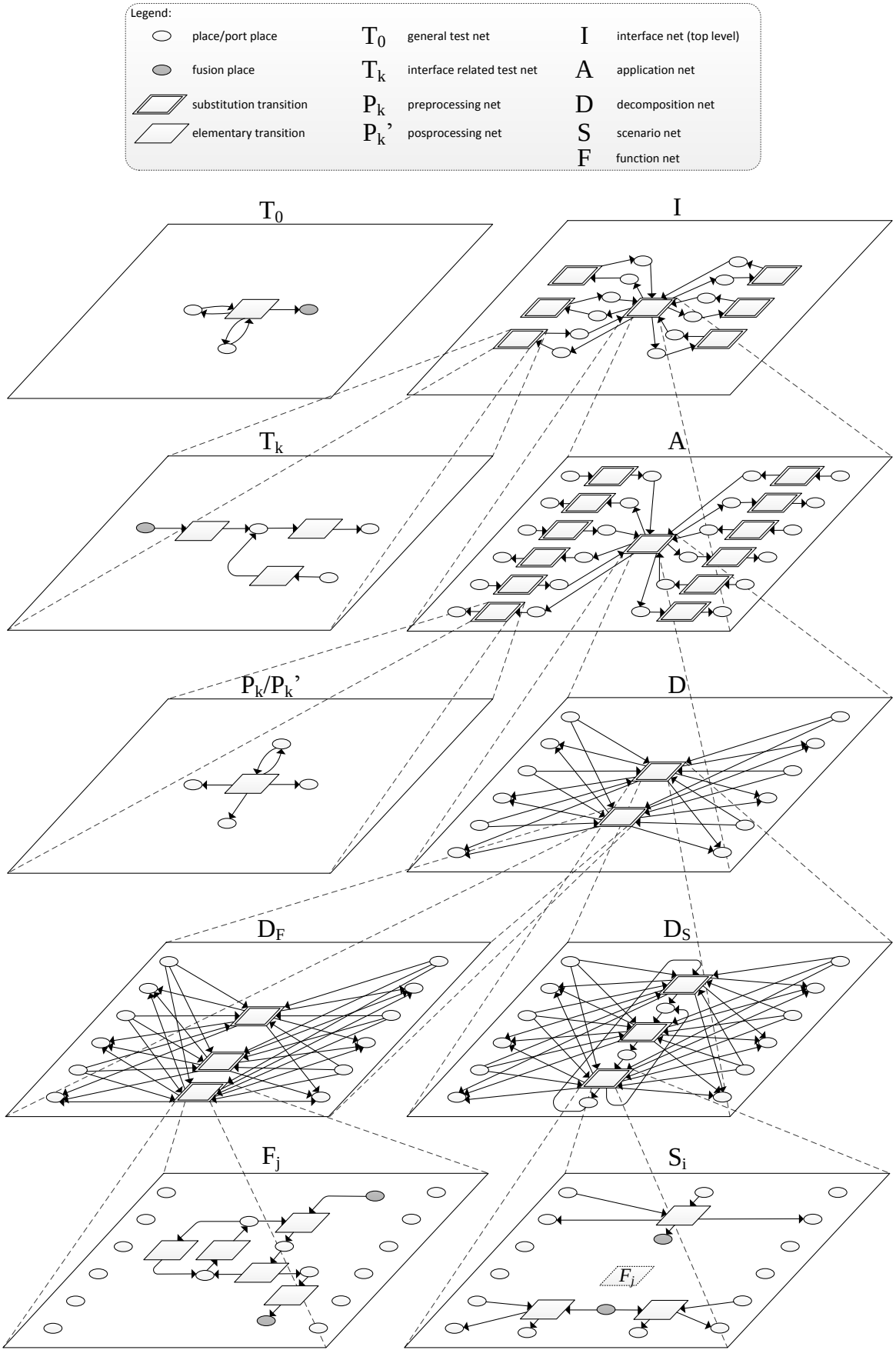


FIGURE 3.5: Decomposition of the main subsystem models [12]

> "The train that in a train station or on a free block section has received a movement authority for the next block section and has/hasn't entered this block section. A release for the following block section is/isn't required."

To illustrate the scenario, UML sequence diagrams can be adopted. A sequence diagram is a kind of interaction diagram that shows the interaction between objects over time. The requirement (scenario RUNNING) presented above can be mapped into the sequence diagram shown in Figure 3.6. There are three objects: `LocUnit`, `Onboard` and `TCC` representing the localisation unit, the on-board subsystem and the traffic control centre, respectively. The diagram illustrates that when the on-board subsystem receives a location data (`loc`) from the localisation unit, a location report will be sent to the traffic control centre immediately. If the value of the location data is equal to a *MA point* stored by means of the on-board map in advance, then a movement request (message `mb1`) will be sent to the traffic control centre. A MA point is a point that when the train passes over, a movement request should be sent to the traffic control centre. Consequently, if the train wants to enter the following block section, a MA for the block section is required.

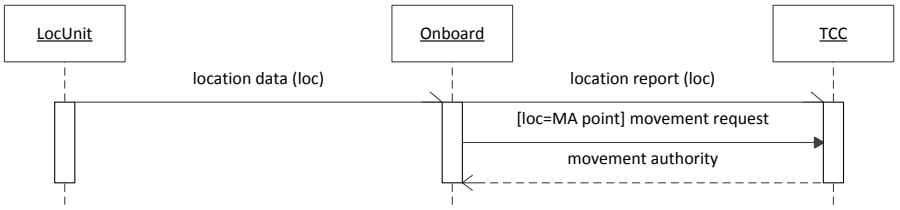


FIGURE 3.6: Sequence diagram for the scenario RUNNING

However, the distinction between the "scenario nets" in the system model and "scenarios" in describing the operational process of the system should be clarified. A *scenario net* S_i of a subsystem model is a *page* [2] that describes a specific process of internal data processing with respect to the incoming data from the environment of the subsystem model. For example, the scenario net `OB_SN_Running` defined in the on-board subsystem model, is a page that shows the process of internal data processing, which is the consequence of the reception of a MA. Nonetheless, in the CPN model of the train control system, the scenarios such as described in Figure 3.6 are realised by exchanging tokens between different scenario nets in different subsystem or component models. Therefore, to apply the generic structure shown

[2] A page is a part of a CP-net that links to other pages through shared places and thereby enables to exchange tokens between pages in different layers.

in Figure 3.5 to a specific system, the method of realising the switching from one scenario net to the others is inevitable. For this purpose, two approaches are proposed to enable the switchovers.

In the first approach (see Figure 3.7), the net D_S shows all the scenario nets that are related to this subsystem and the possible paths of switching the subsystem from one scenario net to the others. In Figure 3.7, the scenario nets related to this subsystem are S_1, S_2, and S_3 represented by substitution transitions. Places `IN` and `OUT` are the input and output ports that are connected to its environment, i.e., the communication subsystem which represents the communication channels between the on-board subsystem and traffic control centre. Places `Flag1, Flag2`, and `Flag3` are the preconditions for the activation of each scenario net. For instance, if `Flag3` is marked, then the scenario net S_3 is activate. Since the subsystem can only stay in one specific scenario net at a certain moment, in other words, the marking of the places `Flag1, Flag2`, and `Flag3` is exclusive, the token on the place `Flag3` will be taken when the activated scenario net of the subsystem model switches from S_3 to S_1 or S_2.

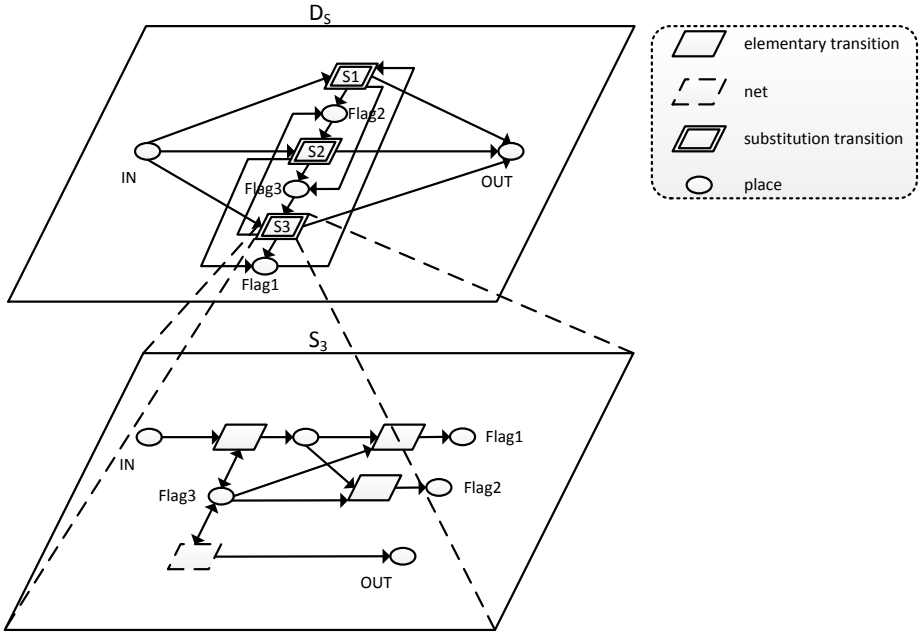


FIGURE 3.7: First approach of switchover of scenario nets

The main idea of the second approach (see Figure 3.8) is that we add an additional net C represented by the substitution transition `C` in net D_S to determine which scenario net is about to be activated. The input of the net C is the received telegrams and the outputs are the preconditions for activating scenario nets. Obviously, only one of the three places `Flag1, Flag2`, and `Flag3` can be marked at one time. It is easy to observe that the second net D_S is much more straightforward than the net D_S in the first approach. Nevertheless, the

first approach is able to show the possible paths of switching the scenario net from one to the others. In the second approach, only the received telegrams can determine the activated scenario net in the next period of time, so in our case, the first approach is preferred. For instance, if a none-defined (unknown) message is received, then there will be such a period of time that no scenario net is activated.

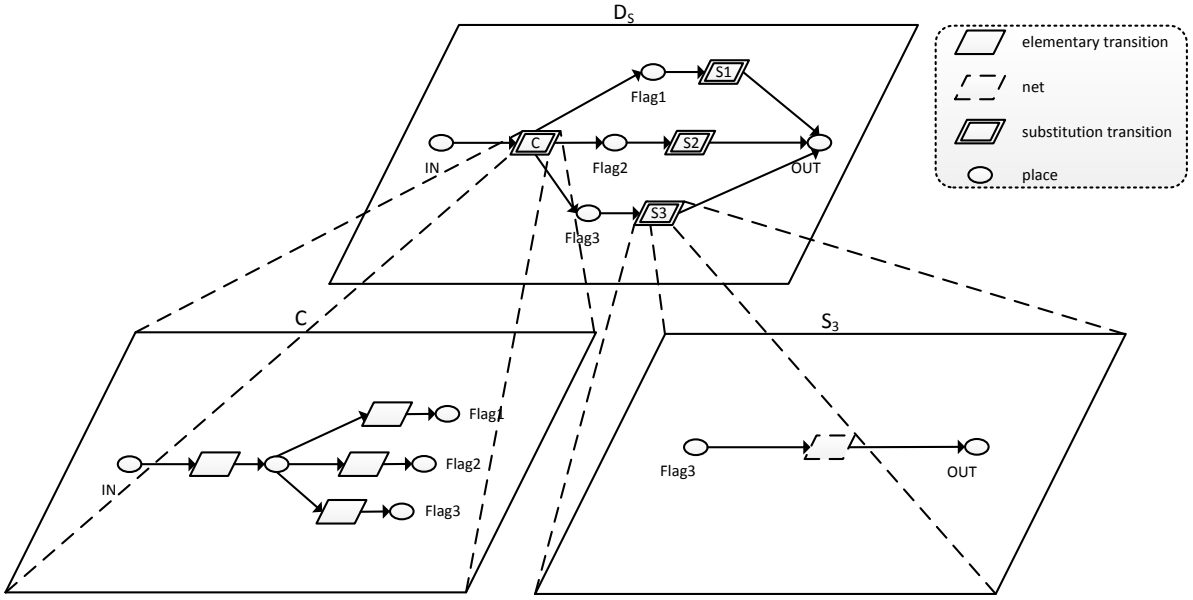


FIGURE 3.8: Second approach of switchover of scenario nets

3.2.5 Module Synchronisation for Quality Assurance

The operation of a concurrent train control system are modelled with CPNs in this work. In addition, the CPN model of the system is also concurrent with regard to the firing of transitions. As mentioned, with the BASYSNET method, the quality assurance for the system model could be carried out by simulation, testing and formal analysis. In general, both the simulation and testing of a Petri net model are realised by model execution and thus share certain common effects on verification of the system model. For formal analysis (model checking), the reachability graph of the Petri net model is usually required to check the reachability property of the model. The simulation of a Petri net model on the computer is in a serial way, that is to say, a model execution corresponds to a firing sequence of transitions. In other words, a simulation process is a path in the reachability graph starting from the initial marking. If there exist circles in the reachability graph, it implies that the firing sequence could be infinite. Figure 3.9 highlights the contradiction between the characteristics of simulation and real system, system model and reachability graph.

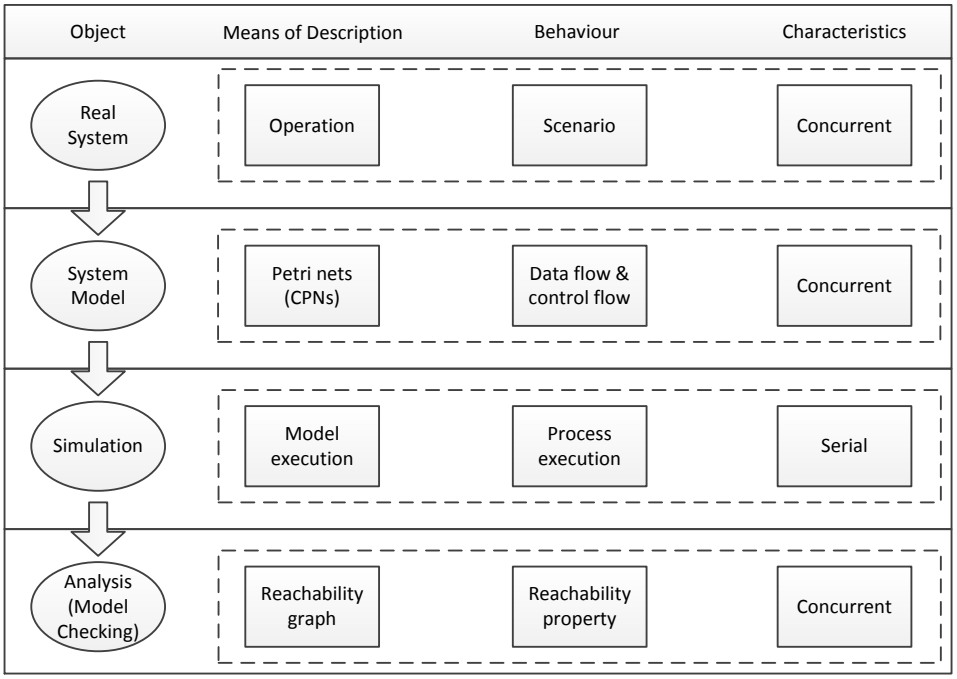


FIGURE 3.9: Four-layer illustration for real system, system model, simulation and analysis

For a system (or an relatively independent subsystem) model, the reachability graph depends on the initialisation of the model. Therefore, the reachability graph of a system model is constant assuming that the initialised data is invariable. When we animate a system model, the input data from the outside environments could be seen as the initialised data of the model. If the model has received an input data, then the model is regarded as having been finished the intended initialisation. However, if a second input data has been received by the model while the previous one is still being processed on the model, then undesired activities and behaviours of the model that may cause difficulties in model verification could occur. The second input data could be seen as a new initialisation of the model, which is out of our intention because this will make the simulation inefficient and non-effective for improving the quality of the model. Hence, controlling the time interval of the inputs of a system model is necessary for continuous simulation. This is done by module synchronisation.

Synchronisation of Transitions. In CPN models, the synchronisation of two modules can be realised by synchronising the key transitions in these two modules. For synchronisation purpose, reference variables defined as the type of `globref` are introduced. A *reference variable* is similar to a pointer in C language. The scope of a reference variable is the entire CPN [35]. In Figure 3.10, pages i and j are two pages (modules) of a CPN model. If there exists a requirement that only the transition t_j in page j has fired, the transition t_{i_2} in page i can fire, then a solution is to introduce the reference variable `gTimer`. The definition of `gTimer`

`(globref gTimer = 0:INT;)` shows that `gTimer` is a type of integer and its initial value is 0. The operation `!gTimer` indicates the contents of the reference variable `gTimer`. The action of transition t_j `(gTimer := !gTimer+1)` means that the value of the integer reference variable `gTimer` will plus one if the transition t_j fires. In addition, fusion places are also used in these pages. Assuming that the transition t_j has never fired in the initial state, then the markings of the places P_F and P_{i_1} are both 0 and the transition t_{i_1} cannot fire under the firing condition of $nGT' > nGT$. Thus the transition t_{i_2} is not fireable. Once t_j has fired, the markings of places P_F and P_{i_1} will be 1 and 0, respectively. Consequently, transitions t_{i_1} and t_{i_2} are fireable in succession, and so on.

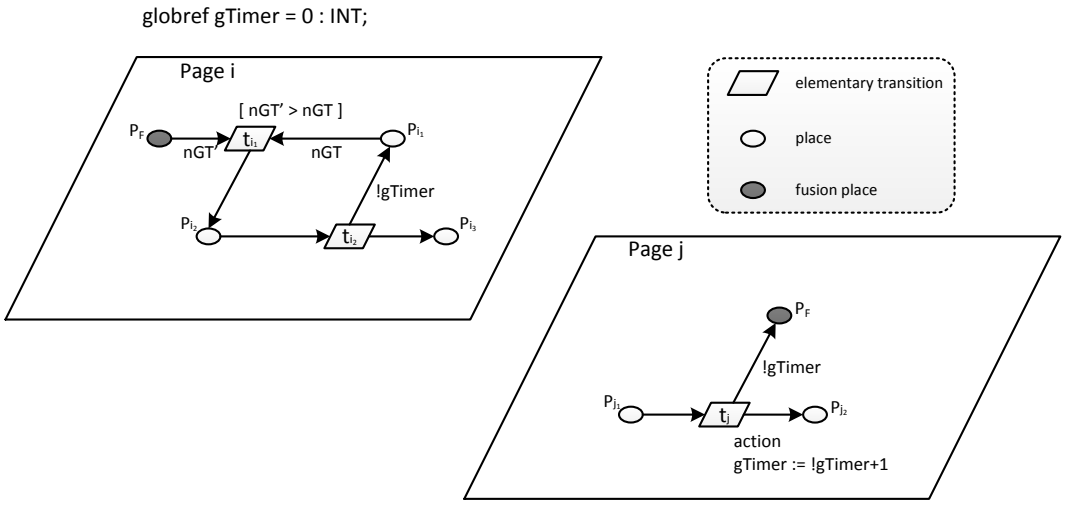


FIGURE 3.10: Synchronisation of two transitions

Synchronisation of the On-board Module and the Module of Localisation Unit. For a satellite-based train control system, the localisation unit is a crucial part of the entire system since there is no other track-side equipment that is used to determine the location of the train. The calculation of the location of the train depends on the localisation unit completely. For the system model, the localisation unit will send location data to the on-board subsystem continuously no matter whether the on-board subsystem has already accomplished processing of the previous ones. We assume that a real-time train control system needs barely no time to process a message, in other words, the system will always had finished processing the previous message when a second one has been received. To imitate this behaviour of the real-time system, continuous simulation needs to synchronise the module of the localisation unit and the on-board subsystem. This is illustrated in Figure 3.11. The scenario net S_i deals with the received location data represented by the place `Location Data` with two processing processes. One is for generating the telegrams that are sent to the traffic control

centre. The other is to generate location reports which are also delivered to the traffic control centre. Both the generated telegrams and the location reports are transmitted by the same communication channel through the interface of place OUT. According to the introduction of synchronisation of transitions above, only if both the generated telegram and the location report are delivered to the place OUT successfully, then the localisation unit can send a next location data to the on-board subsystem.

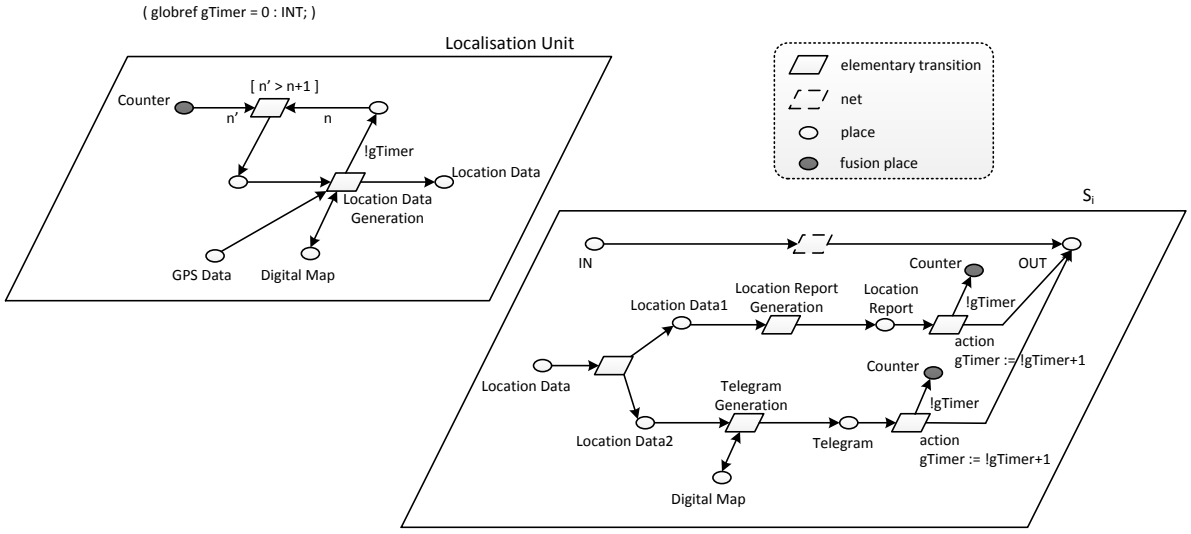
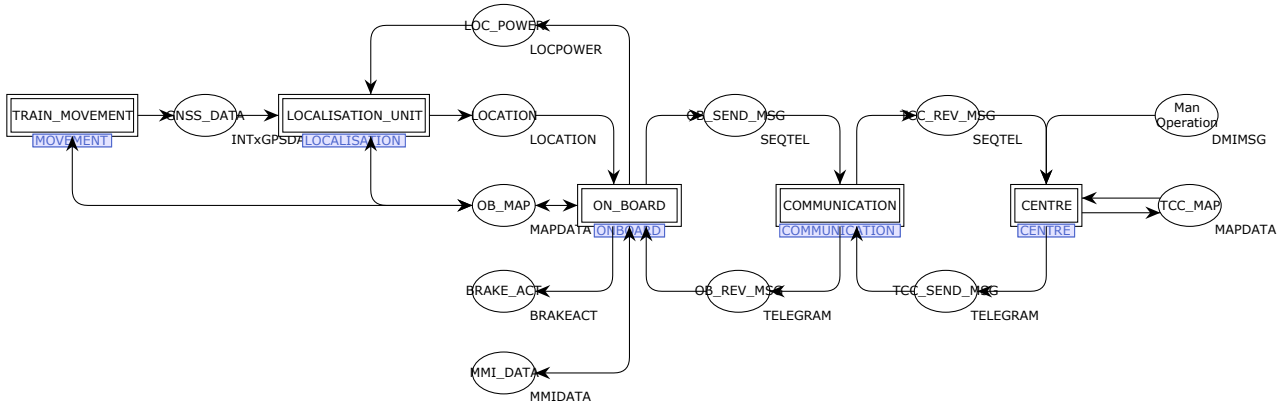


FIGURE 3.11: Synchronisation of the on-board module and the module of the localisation unit

3.3 Application Example: the On-board Module of SatZB Model

Following the modelling approach presented forwards, a first CPN model of SatZB has been developed according to the functional requirements specification. The system model has two major modules ON_BOARD and CENTRE corresponding to the on-board subsystem and the traffic control centre, respectively. For the purpose of separate development, the localisation unit (LOCALISATION_UNIT) is developed as an independent subsystem instead of an unit of the on-board subsystem. The communication system between the on-board subsystem and the traffic control centre is also seen as an independent subsystem. The module TRAIN_MOVEMENT imitates the movement of the train so as to generate dummy GNSS data. The architecture (top level) of the CPN model is shown in Figure 3.12. Tokens on the places represent the data or messages exchanged between the subsystems. Since the on-board subsystem and the traffic control centre are the two main subsystems of SatZB and the modelling

approach, i.e, the vertical decompositions of the subsystem models, are the same, the on-board module of SatZB model is presented in this section as the application example.



1`{sequence = 1, ID1=1,LAT1="3599679.5513",LONG1="5797738.8174",HEIGHT1="78.4",LAT2="3599702.1687",LONG2="5797719.8690",HEIGHT2="78.4",MILEAGE1="0",LENGTH12="29"}++
1`{sequence = 2, ID1=0,LAT1="3599702.1687",LONG1="5797719.8690",HEIGHT1="78.4",LAT2="3599734.9683",LONG2="5797687.2684",HEIGHT2="77.5",MILEAGE1="29",LENGTH12="46"}++
1`{sequence = 3, ID1=0,LAT1="3599734.9683",LONG1="5797687.2684",HEIGHT1="77.5",LAT2="3599766.9168",LONG2="5797659.6596",HEIGHT2="76.7",MILEAGE1="75",LENGTH12="42"}++
1`{sequence = 4, ID1=2,LAT1="3599766.9168",LONG1="5797659.6596",HEIGHT1="76.7",LAT2="3599797.6459",LONG2="5797634.4381",HEIGHT2="76.5",MILEAGE1="117",LENGTH12="39"}++
1`{sequence = 5, ID1=0,LAT1="3599797.6459",LONG1="5797634.4381",HEIGHT1="76.5",LAT2="3599824.5652",LONG2="5797610.9022",HEIGHT2="76.4",MILEAGE1="157",LENGTH12="35"}++
1`{sequence = 6, ID1=3,LAT1="3599824.5652",LONG1="5797610.9022",HEIGHT1="76.4",LAT2="3599858.0133",LONG2="5797582.3040",HEIGHT2="76.5",MILEAGE1="193",LENGTH12="44"}++
1`{sequence = 7, ID1=0,LAT1="3599858.0133",LONG1="5797582.3040",HEIGHT1="76.5",LAT2="3599887.8692",LONG2="5797555.8781",HEIGHT2="76.7",MILEAGE1="237",LENGTH12="39"}++
1`{sequence = 8, ID1=4,LAT1="3599887.8692",LONG1="5797555.8781",HEIGHT1="76.7",LAT2="3599920.0571",LONG2="5797530.5756",HEIGHT2="76.7",MILEAGE1="277",LENGTH12="40"}++
1`{sequence = 9, ID1=0,LAT1="3599920.0571",LONG1="5797530.5756",HEIGHT1="76.7",LAT2="3599946.0076",LONG2="5797507.7627",HEIGHT2="77.2",MILEAGE1="318",LENGTH12="34"}++
1`{sequence = 10, ID1=5,LAT1="3599946.0076",LONG1="5797507.7627",HEIGHT1="77.2",LAT2="3599946.0076",LONG2="5797507.7627",HEIGHT2="77.2",MILEAGE1="352",LENGTH12="0"}

FIGURE 3.12: The top level of the CPN model of SatZB

3.3.1 Declarations

The declarations of colour sets and variables that associated with the on-board subsystem model are presented as following:

```
colset INT = int; (*integer data*)
colset STRING = string; (*string data*)
colset BOOL = bool; (*boolean data*)
colset LOCPOWER = BOOL;
colset MAPDATA = record sequence:INT*
                        ID1:INT*
                        LAT1:STRING*
                        LONG1:STRING*
                        HEIGHT1:STRING*
                        LAT2:STRING*
                        LONG2:STRING*
                        HEIGHT2:STRING*
                        MILEAGE1:STRING*
                        LENGTH12:STRING; (*map data*)
colset DIRECTION = with UP | DOWN | UNKNOW; (*train direction*)
```

```
colset POS = STRING; (*position offset (modified)*)
colset DIR = DIRECTION;
colset LOCATION = product POS * DIR; (*train location*)
colset MMIDATA = STRING; (*MMI data*)
colset BRAKEACT = STRING; (*brake*)
colset TELEGRAM = record TRAINID:TRAINIDNUMBER* (*train ID*)
                         MSGID:MSGIDNUMBER* (*message ID*)
                         TSTAMP:TIMESTAMP* (*time stamp*)
                         DATA:STRING; (*message content*)
colset SEQTEL = product INT * TELEGRAM;
colset BLOCKPOINT = product INT * STRING; (*block point*)
colset BLOCKPOINTLIST = list BLOCKPOINT;
colset STRINGLIST = list STRING;

var msg,msg':TELEGRAM;
var map:MAPDATA;
var seqtel:SEQTEL;
var listBlockPoint:BLOCKPOINTLIST;
var sList,sList1,sList2:STRINGLIST;
var n,n',t,t',num,seq:INT;
var p,p',m,m',r,ir,brk,str,str':STRING;
var loc,loc':LOCATION;
var fmmi,tmmi:MMIDATA;
var b,mapvr,be,bl,breg:BOOL;
var d,d':DIRECTION;

fun check(x,l) = (mem l x); (*check if x is a member of the list l*)
globref gTime = 0:INT; (*reference variable*)
```

Note that $check(x, l)$ is a function that checks whether x is member of the list l and $gTime$ is a reference variable initialising by 0.

3.3.2 Submodules

According to the functional requirements specification of SatZB, seven scenarios are specified: INITIALISATION, REGISTRATION, RUNNING, CONDITIONAL RUNNING, BAN OF ENTRY, EMERGENCY STOP and LOGOUT (note that another scenario SHUNTING is not considered in this work). Accordingly, seven corresponding scenario nets (i.e., submodules `OB_SN_Initialisation`, `OB_SN_Registration`, `OB_SN_Running`, `OB_SN_Conditional_Running`, `OB_SN_Ban_Of_Entry`, `OB_SN_Emergency_Stop` and `OB_SN_Logout`) are defined in the on-board module of SatZB model shown in Figure 3.13, which is the submodule of the substitution transition `ON_BOARD` in Figure 3.12. This module refers to the combination of the nets D_F and D_S in Figure 3.26 so that reduces the net D. Submodules `OB_FN_EBrake_Activation` and `OB_FN_NBbrake_Activation` are two function nets.

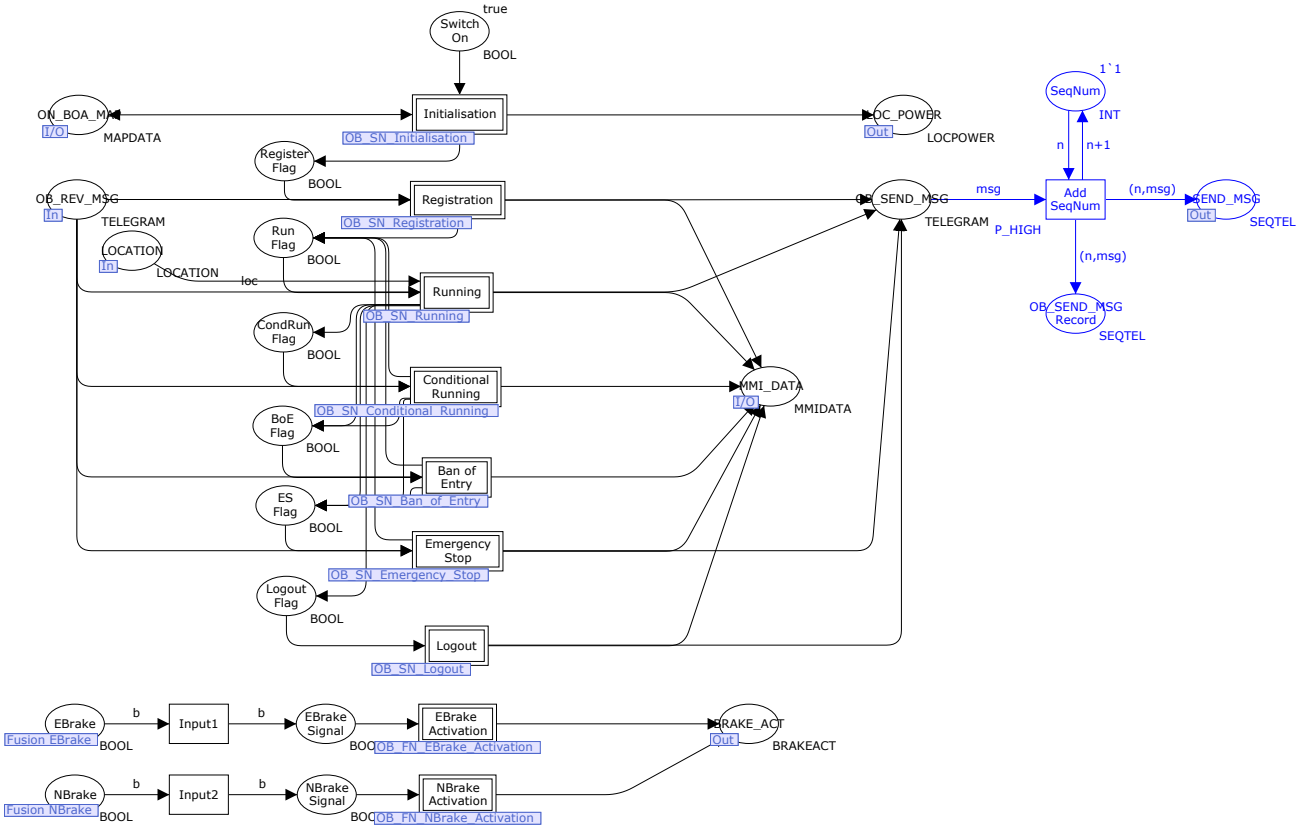


FIGURE 3.13: The on-board module

Places `LOCATION` and `OB_REV_MSG` are input channels and the markings (tokens) of these places represent messages that can be received by the on-board subsystem but not yet received. Places `OB_SEND_MSG`, `LOC_POWER` and `BRAKE_ACT` are output channels and the markings of these places represent messages or signals that are about to be sent out by the on-board subsystem but not yet sent out. Places `OB_MAP` and `MMI_DATA` are input/output channels that can both receive and send data. Places `EBrake` and `NBrake` are fusion places that as the inputs of the function nets which outputs are directly delivered to the output

TABLE 3.2: Messages transmitted from train to TCC

Message	Name	Remarks
mb1	movement request	
mb2	notification of entry	
mb3	notification of leaving	
mb4	emergency stop demand	
mb5	request to cancel the state of emergency stop	
mb6	request for shunting operation	out of consideration
mb7	request to end of shunting operation	out of consideration
mb8	registration request	
mb9	logout request	

TABLE 3.3: Messages transmitted from TCC to train

Message	Name	Remarks
mt1	movement authority	
mt2	deny of movement	
mt3	conditional movement authority	
mt4	command for emergency stop	
mt5	release of emergency stop	
mt6	authority for shunting operation	out of consideration
mt7	authority to end of shunting operation	out of consideration
mt8	registration data	
mt9	logout authority	

channel `BRAKE_ACT`. Transition `Add SeqNum` with a high firing priority is added to convert the sending messages to message sequences by adding a sequence number for each message. Places `Switch On`, `Register Flag`, `Run Flag`, `CondRun Flag`, `BoE Flag`, `ES Flag`, and `Logout Flag` are the preconditions for activating the corresponding scenario nets. From the Figure 3.13 it is easy to observe that the switchovers of scenario nets can be illustrated by Figure 3.14. The messages transmitted between the on-board module and the model of the traffic control centre are listed in Table 3.2 and Table 3.3.

Submodule OB_SN_Initialisation. Figure 3.15 shows the submodule of the scenario net `OB_SN_Initialisation`. The main task of this net is to identify the *MA (movement authority) points*, *section points*, and *logout points* according to the on-board map. A MA point is a point of the rail track that when the train reaches it, a movement request for the next block section should be sent to the traffic control centre; section points are the borders of the block sections; logout points are the points that the train is allowed to log out and switch off. After identifying these points, places `LOC_POWER` and `Register Flag` will be marked by firing the transition `Output Data` which has a low firing priority.

Submodule OB_SN_Registration. Figure 3.16 shows the submodule of the scenario net `OB_SN_Registration`. When the on-board module is successfully initialised (the place `Register`

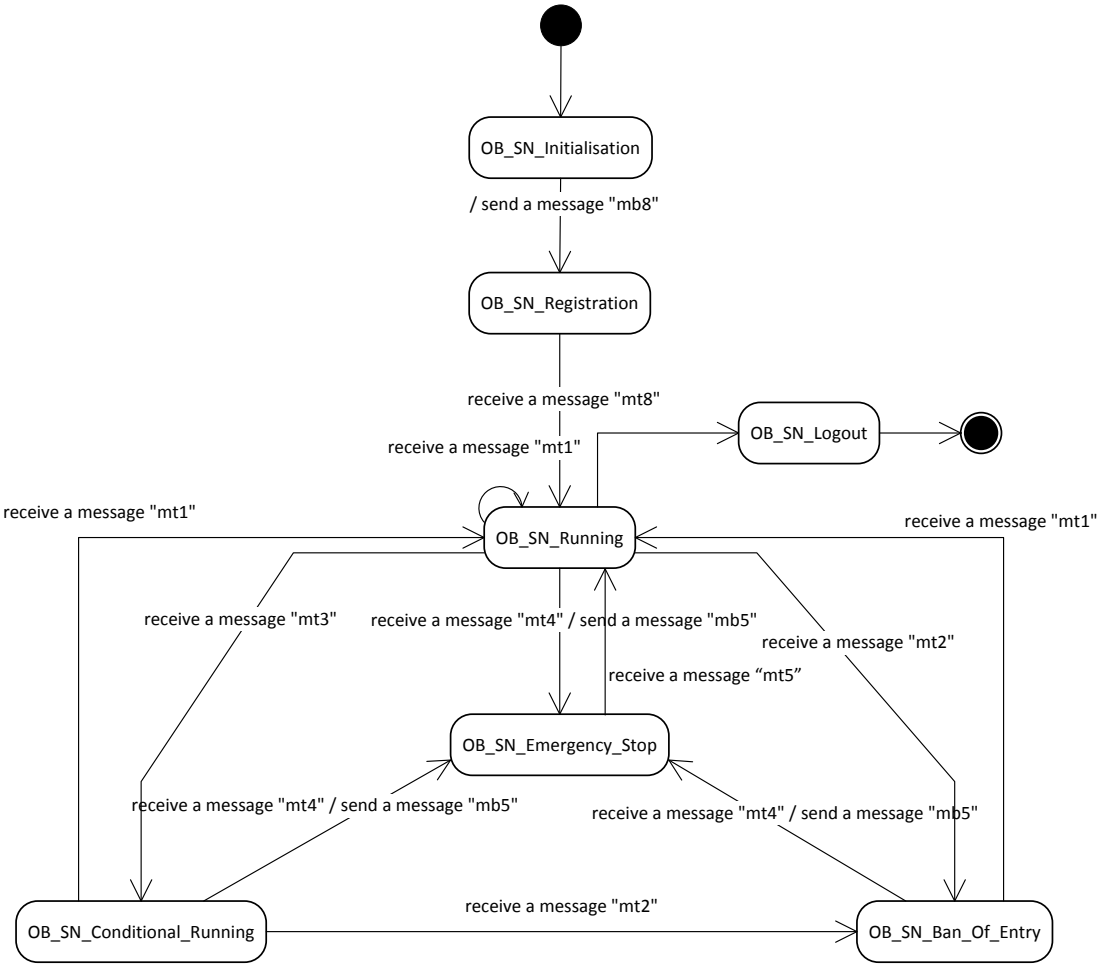


FIGURE 3.14: State diagram for the transitions of scenario nets

`Flag` is marked), it means that the underlying system, the on-board subsystem has turned into the scenario REGISTRATION. And then a registration request whose message ID is “mb8” (For convenience, the request is called message `mb8`, so do other messages.), will be sent to the traffic control centre and related texts will be displayed on the MMI (Man Machine Interface). If a message `mt8` is received afterwards, then the token on the place `Register Flag` will be removed and the place `Run Flag` will be marked by firing the transition `Running`, which indicates that the on-board subsystem has switched to the scenario RUNNING.

Submodule OB_SN_Running. Figure 3.17 shows the submodule of the scenario net `OB_SN_Running`. When a location data is received by the net, it is compared with the MA points, section points and logout points. If the location data matches a MA point, then a movement request `mb1` along with a location report, will be sent to the traffic control centre; if the location data matches a section point, then two consecutive notification messages `mb2` (the train has entered the block section.) and `mb3` (the train has left the block section.) along with a location report will be sent to the traffic control centre sequentially; if the location data matches a

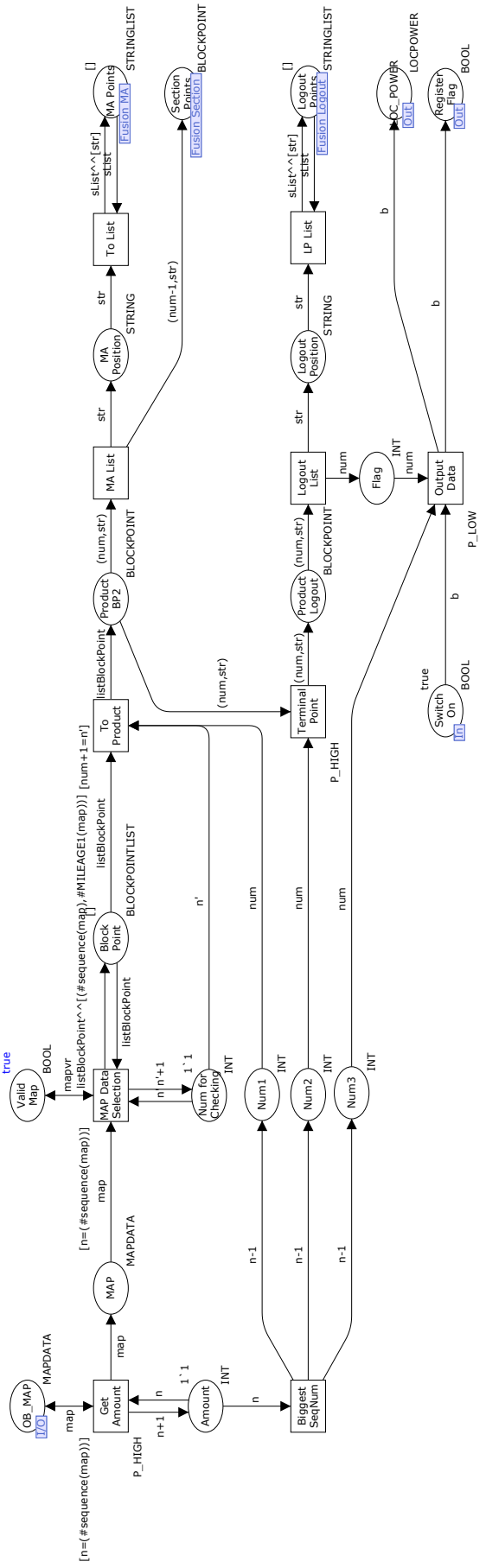


FIGURE 3.15: Submodule `OB_SN_Initialisation`

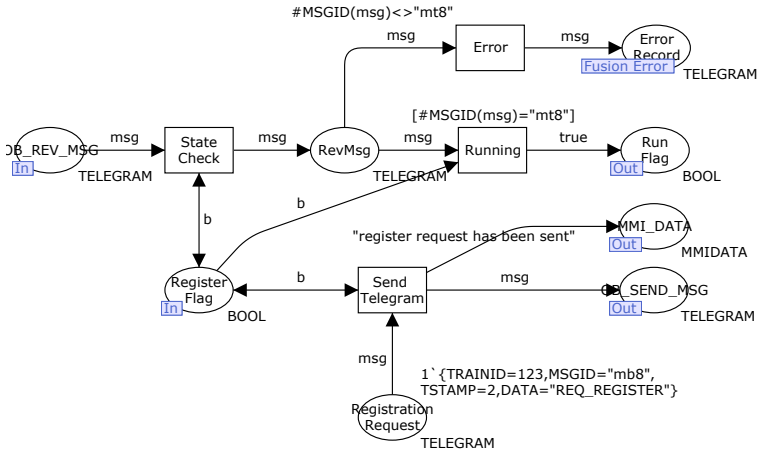


FIGURE 3.16: Submodule `OB_SN_Registration`

logout point, then a logout request `mb9` along with a location report, will be sent to the control centre; otherwise, only a location report will be sent to the traffic control centre. Location reports are generated through the submodule (see Figure 3.18) represented by the substitution transition `Location report`.

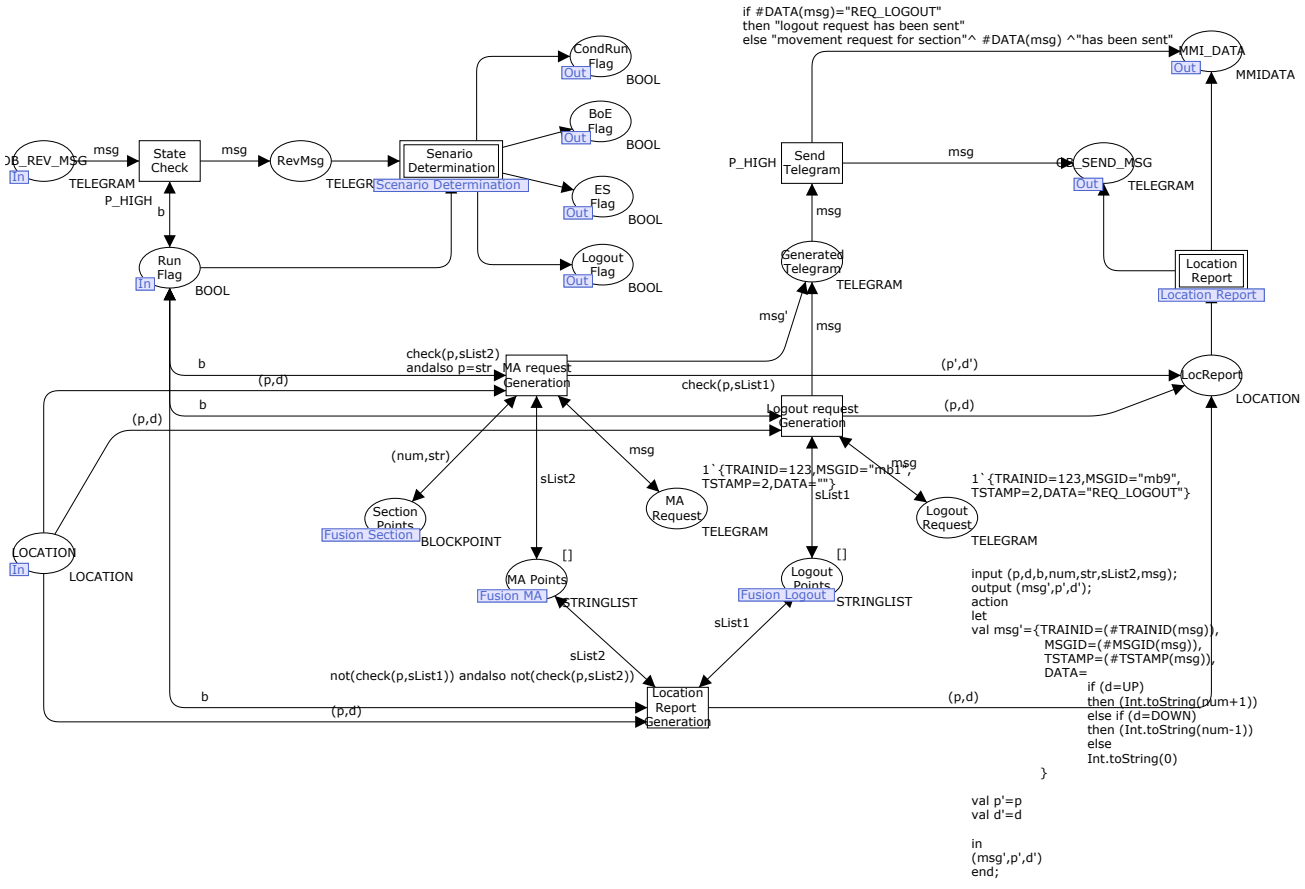


FIGURE 3.17: Submodule `OB_SN_Running`

In Figure 3.18, when a location data is received by the place `LocReport`, it is compared with the section points, MA points and logout points. If it matches a section point as well

as a MA point or a logout point, and a movement authority for running has been received previously (if so, place `Movement Authority` will be marked), then a message `mb2` will be sent to the traffic control centre following by a message `mb3`. The fusion place `gTime` is used to synchronise the modules `ON_BOARD` and `LOCALISATION_UNIT`. It is observed that only the message `mb3` has been sent out or the transition `Discard LocReport` has been fired, the reference variable `gTime` will plus one, which makes the module `LOCALISATION_UNIT` send a next location data to the on-board module.

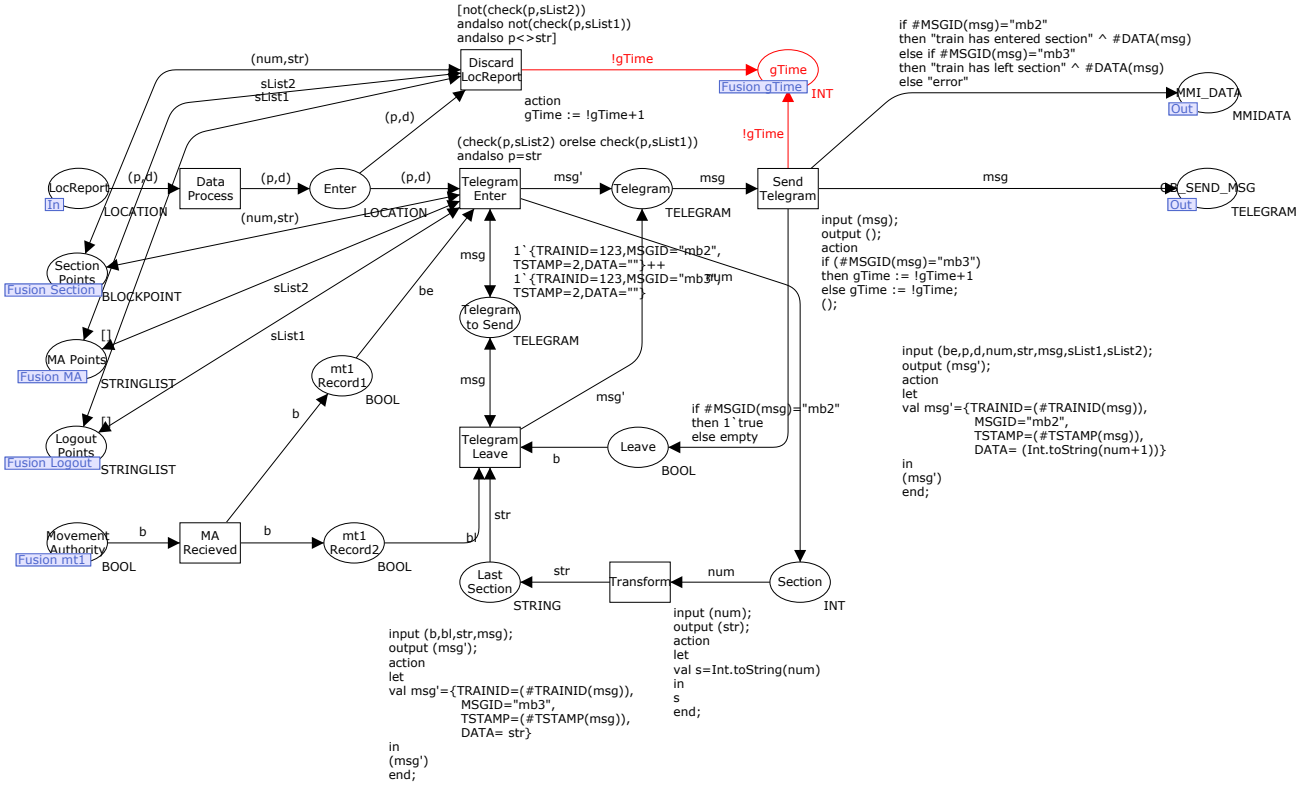


FIGURE 3.18: Submodule `Location Report`

When a message from the traffic control centre is received by the net `OB_SN_Running`, it is checked with the submodule (see Figure 3.19) represented by the substitution transition `Scenario Determination`. In Figure 3.19, when the message received from the traffic control centre is `mt2`, `mt3`, `mt4` or `mt9`, the token on the place `Run Flag` will be taken and one of the places `BoE Flag`, `CondRun Flag`, `ES Flag` or `Logout Flag` will be marked. Which one to be marked is determined by the ID of the received message. Besides, if the received message is `mt2` or `mt4`, then the fusion place `NBrake` or `EBrake` will be marked, which indicates the calling for the function nets `OB_FN_NBrake Activation` (normal brake, see Figure 3.20) and `OB_FN_EBrake Activation` (emergency brake, see Figure 3.20), respectively.

When a message from the traffic control centre and a location data from the localisation unit are simultaneously received by the input channels of the places `OB_REV_MSG` and `LOCATION`,

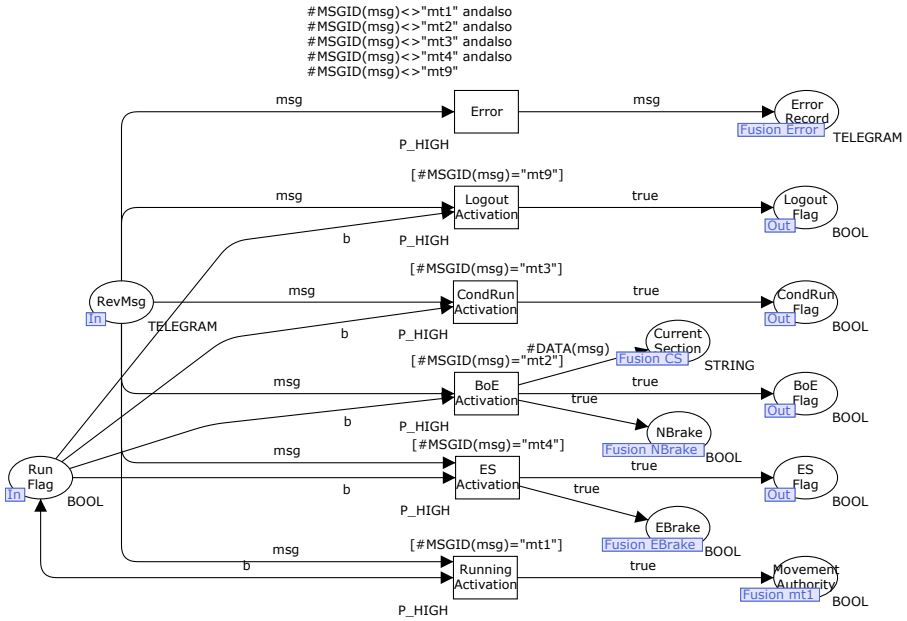


FIGURE 3.19: Submodule `Scenario determination`

the processing of the message from the traffic control centre has a higher priority. Therefore, the transition `State Check` and the transitions in Figure 3.19 are designated as having a higher firing priority (inscribed with the `P_HIGH`). The declarations of firing priorities of transitions are defined as follows.

```
val P_HIGH = 100;
val P_NORMAL = 1000;
val P_LOW = 10000;
```

Transitions without any inscription of firing priorities have normal priority. When two transitions, one is inscribed with the `P_HIGH` and the other one is inscribed with the `P_LOW` or is not inscribed with any firing priority, are enabled at the same time, then the transition inscribed with the `P_HIGH` will fire first.

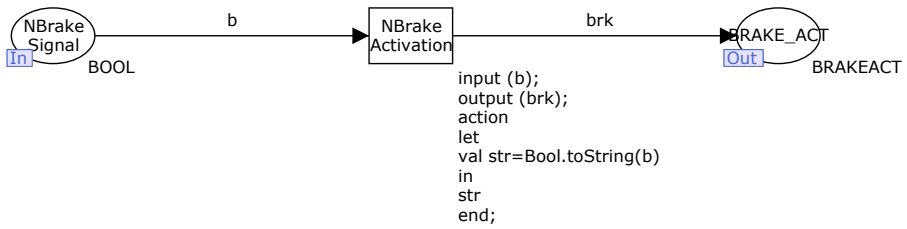


FIGURE 3.20: Function net `NBrake Activation`

Submodule OB_SN_Conditional_Running. Figure 3.22 shows the submodule of the scenario net `OB_SN_Conditional_Running`. When the activated scenario net of the on-board module is switched into this net, a text "conditional running" will be repeatedly delivered to the MMI

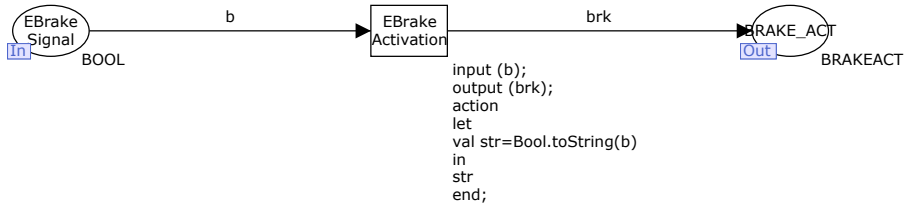


FIGURE 3.21: Function net `EBrake Activation`

until a message `mt1`, `mt2` or `mt4` is received. If the message `mt2` is received, then the fusion place `Current Section` will be marked and the token colour (natural number) on this place indicates the current section of the train.

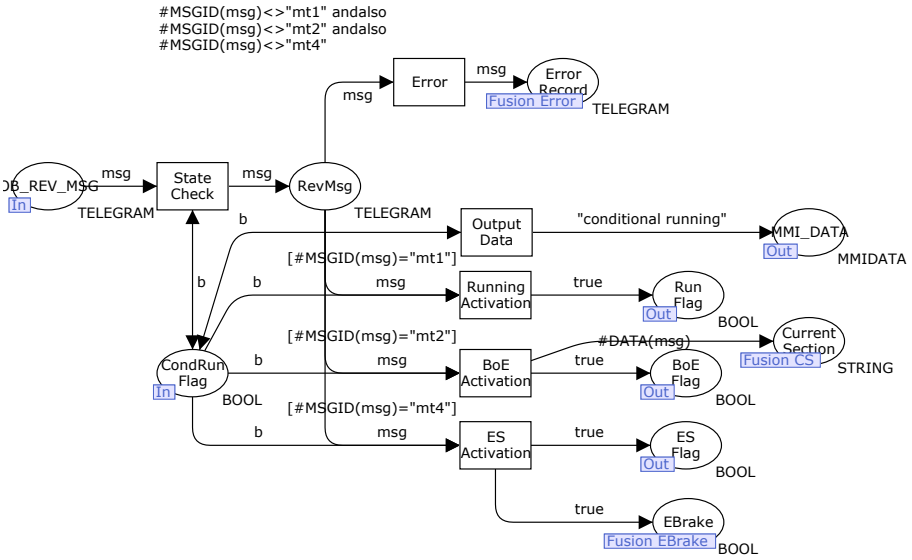


FIGURE 3.22: Submodule `OB_SN_Conditional_Running`

Submodule OB_SN_Ban_Of_Entry. Figure 3.23 shows the submodule of the scenario net `OB_SN_Ban_Of_Entry`. When the activated scenario net of the on-board module is witched into this net, a text "Ban of Entry for section i" will be repeatedly delivered to the MMI until a message `mt1` or `mt4` is received, where i is the identifier of the block section.

Submodule OB_SN_Emergency_Stop. Figure 3.24 shows the submodule of the scenario net `OB_SN_Emergency_Stop`. When the activated scenario net of the on-board module is switched into this net, a request for cancelling emergency stop will be repeatedly sent to the traffic control centre if the conditions for the cancellation, represented by the place `Cancellation Ack`, is fulfilled. When a message `mt5` is received, the on-board module will inactivate this net, i.e., the token on the place `ES Flag` will be taken.

Submodule OB_SN_Logout Figure 3.25 shows the submodule of the scenario net `OB_SN_Logout`. When the activated scenario net of the on-board module is switched into this net, a text "logout" will be delivered to the MMI and a notification message will be sent to the traffic control centre.

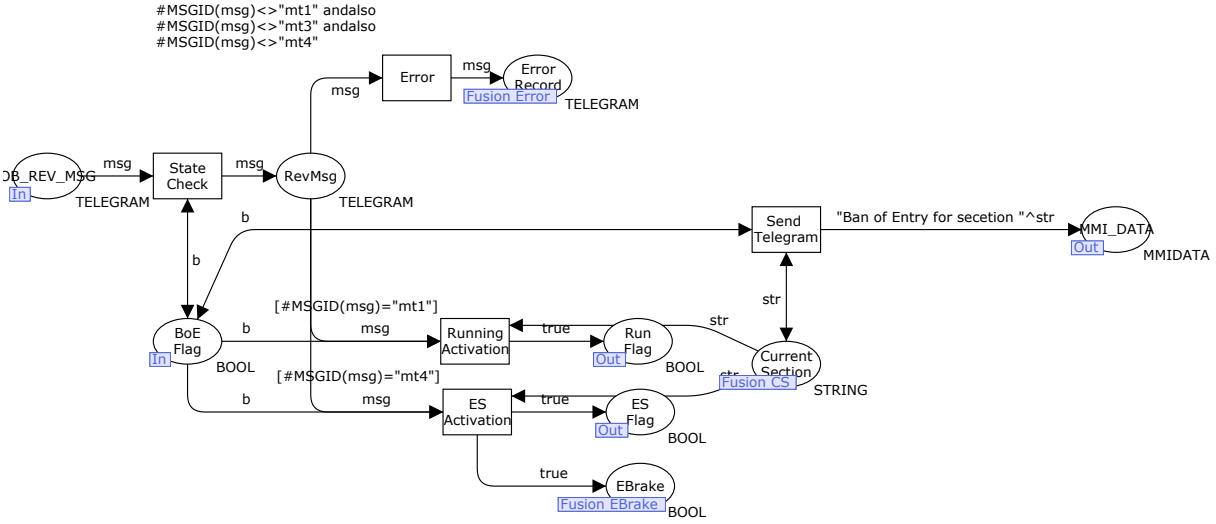


FIGURE 3.23: Submodule `OB_SN_Ban_Of_Entry`

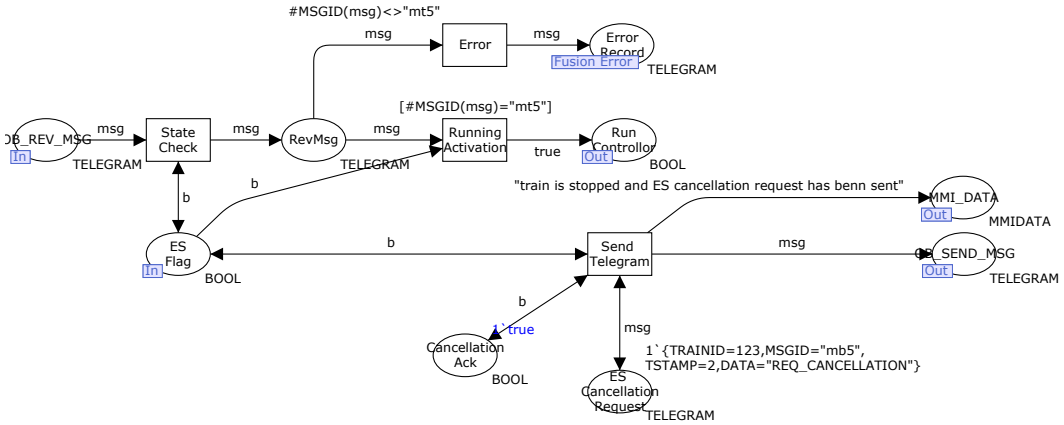


FIGURE 3.24: Submodule `OB_SN_Emergency_Stop`

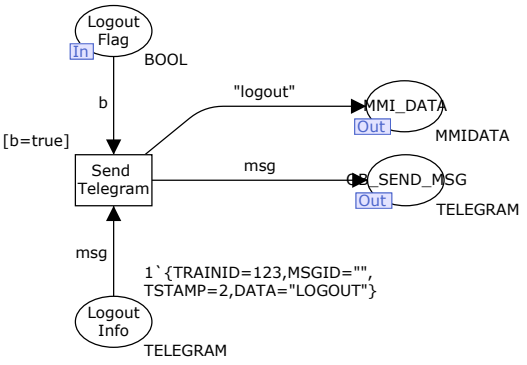


FIGURE 3.25: Submodule `OB_SN_Logout`

3.3.3 Module Composition

The relationship between modules in a hierarchical model can be represented as a directed graph which has a node for each module and an arc for each substitution transition. For the CPN model in Figures 3.12 - 3.25, the *module hierarchy* is shown in Figure 3.26. The names of the modules have been written inside the nodes, and the arcs have been labelled with the names of the substitution transitions. The node has no incoming arcs is a root of the module hierarchy and is called a *prime module*. Node `SatZB` has five outgoing arcs, corresponding to the three substitution transitions in the SatZB module (see Figure 3.12). For instance, the arc from `SatZB` to `ONBOARD`, labelled `ON_BOARD`, specifies that the substitution transition `ON_BOARD` in the `SatZB` module has the module `ONBOARD` as its related module. Note that this chapter focus on the `ONBOARD` module in the `SatZB` module.

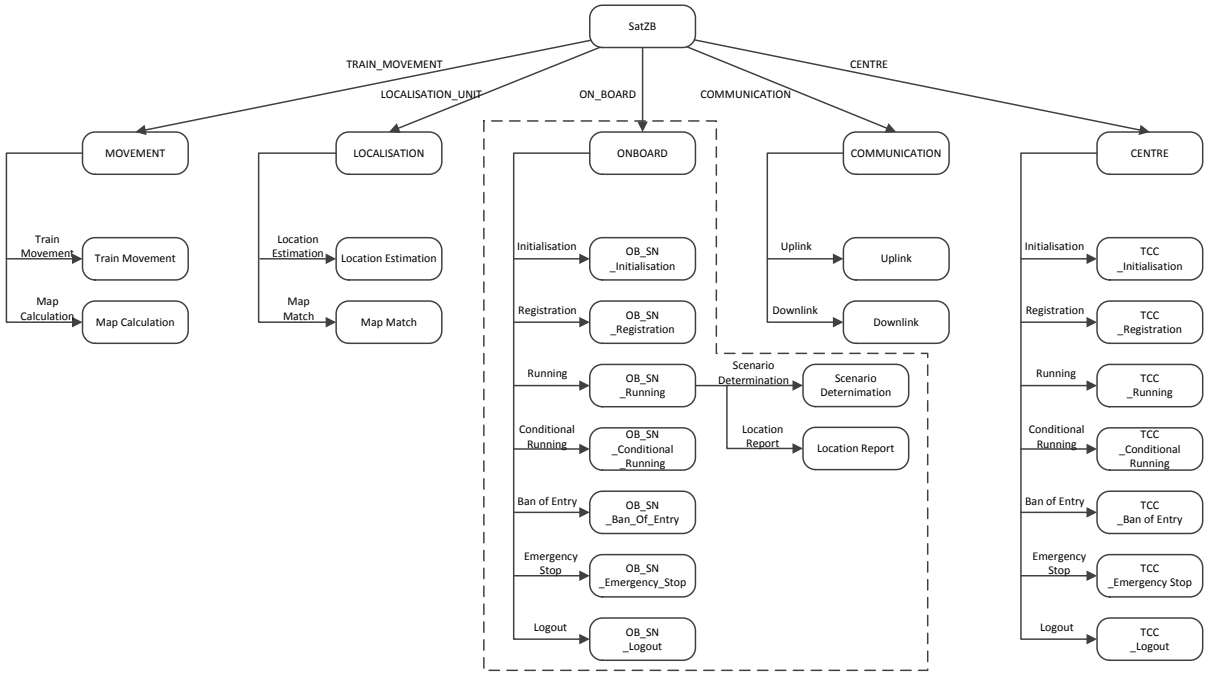


FIGURE 3.26: Module hierarchy for the hierarchical CPN model of SatZB

3.4 Summary

This chapter proposes the approach for realistic modelling of a satellite-based train control system (SatZB). Based on the model architecture presented in [12], a first CPN model of the on-board subsystem is elaborated. In particular, the implementation of switchovers of scenario nets and module synchronisation are emphasised.

In future, the time constrains could be added to the existing model and further refinement could be carried out according to the system requirements specification.

Chapter 4

Identification of Verification Tasks

After establishing a system model, it is important to know what kinds of functions, behaviours or properties need to be verified. This is especially vital for a safety-critical system. Therefore, the verification tasks of the SatZB model developed in chapter 3 will be identified in this chapter.

In this chapter, first the hazard analysis for SatZB system is carried out and the hazardous conditions of SatZB model are identified. And then the functions of SatZB both for safety and normal operation and their allocations in the SatZB model are presented. Last, the verification tasks for the on-board module of the SatZB model are specified.

4.1 Hazard Analysis

In railway domain, risk analysis for a safety-critical system may need to be repeated at several stages of the life-cycle according to the standard EN 50126 [22]. For instance, risk analysis following the phases of concept, and system definition and application conditions aims at identifying hazards associated with the system, and determining the risk associated with the hazards as well as establishing a process for ongoing risk management. Followed by the risk analysis phase, the system requirements including safety requirements are specified. In the design and implementation phase, hazard analysis and risk assessment need to be performed for implementing safety plan. No matter in which phase the hazard analysis is performed, an identification of hazards is required as the basis. In this section, we conduct the hazard analysis with Generic Hazard List (GHL) [99], [100], [101].

4.1.1 Hazard Definition

In standard EN 50126, hazard is defined as "a physical situation with a potential for human injury". This definition expresses a state from which damage or loss may be suffered due to, for example, an accident, but it does not consider the causes of leading the system into this state. In order to include the causes of the physical situation in the definition of hazard, a hazard can be defined by a Petri net model depicted in Figure 4.1 according to [100] and [101]. The Petri net model in Figure 4.1 shows the causal dependencies between human injury (harm), the physical situation (hazardous state) and the causes leading to this situation (hazardous conditions). Therefore, the hazard can be defined as:

Definition 4.1 (Hazard)**.** The defined hazardous conditions and the state transition leading to a hazardous state with a potential of human injury (harm) due to an accident (harmful event) [99].

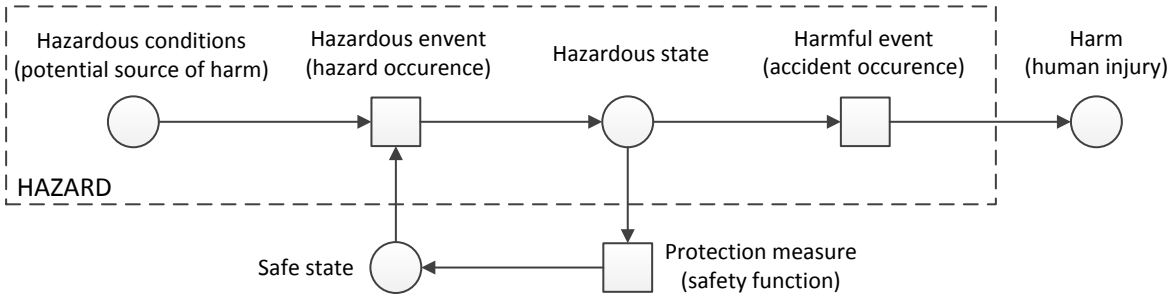


FIGURE 4.1: Definition of hazard described with Petri nets

4.1.2 Hazard Identification

Accident Classification. As it is shown in Figure 4.1, the hazardous state has a close relationship to the accident that may occur. Depending on the defined accident characteristics, hazards can be identified based on the functionality analysis of the system. A classification of accident characteristics in railway from the view of a train control system are illustrated in Figure 4.2 according to [88], [99] and [100]. In Figure 4.2, accidents are classified as two categories: internal accidents that are caused by internal faults of the train control system and external accidents that are resulted from external influences. The external influences cannot be eliminated by doing something about the train control system, but must be identified or detected by the train control system and appropriate reactions have to be made by the system in time. The internal accidents however can be controlled by the train control system

directly. For example, by controlling the speed of the train, some hazardous situations could be prevented from occurring directly.

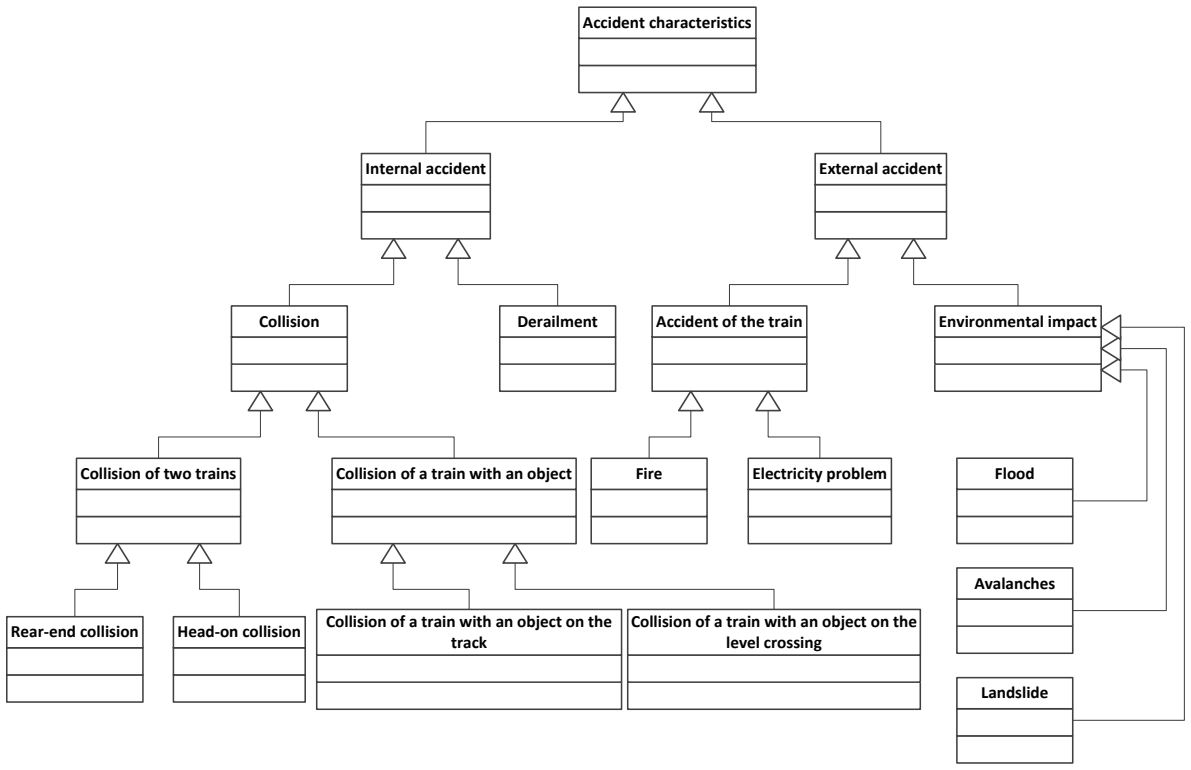


FIGURE 4.2: Classification of accidents [88], [99], [100]

System Structure Analysis. Using the hazard identification approach proposed in [100], [101] and [99], an analysis of the system structure and the system functionality is required. The structure of the SatZB system and its CPN model have been shown in Figure 1.1 and Figure 3.4, respectively.

System Functional Analysis. For the functional analysis of SatZB, an operational process (see Figure 4.3) consisting of functional blocks of the system is considered as the basis. The process shows the data flow of the system and the functions of the system components as well as their causal dependencies. In the sense of hazard analysis, each function of the system components has a potential of being failed, which leads to hazardous states.

Hazard Table. In practice, hazard tables are usually used to describe hazards. In Table 4.1, there are four main areas: functionality area is filled with the functions defined in the operational process (Figure 4.3), accident characteristics area describes the considered classes of accidents (Figure 4.2), resources area is specified on the basis of the system structure and identified hazards area indicates the possible accidents due to false functions of the resources.

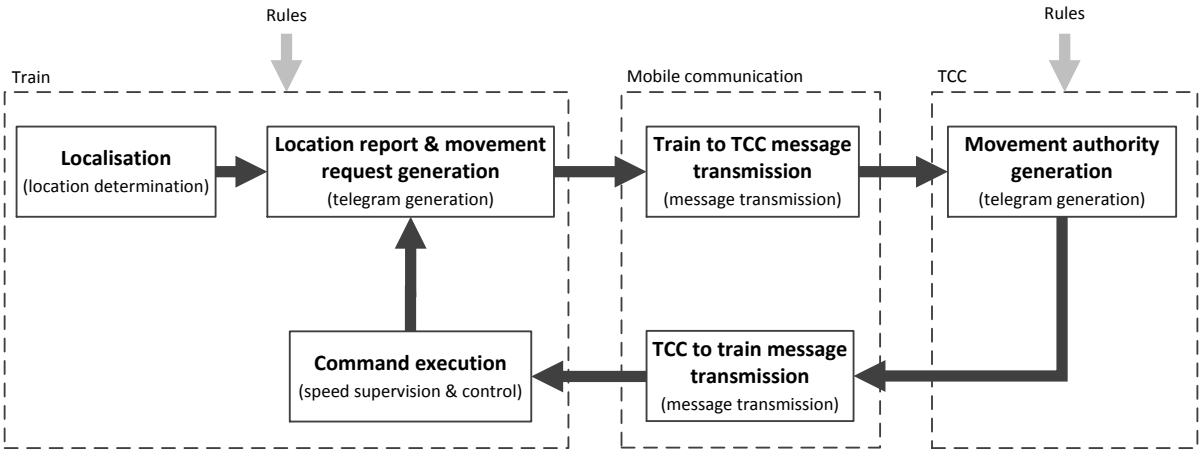


FIGURE 4.3: Operational process of the SatZB system

Since the derailment of a train is often caused by the exceeding of its allowed speed, only the accident characteristics of collision is considered in this work.

Identified Hazards:

A1: Possible collision of two trains due to the generation of *incorrect location data* by the train.

A2: Possible collision of a train with an object due to the generation of *incorrect location data* by the train.

B1: Possible collision of two trains due to the *incorrect execution of commands* by the train.

B2: Possible collision of a train with an object due to the *incorrect execution of commands* by the train.

C1: Possible collision of two trains due to the generation of an *incorrect location report* by the train.

C2: Possible collision of a train with an object due to the generation of an *incorrect location report* by the train.

D1: Possible collision of two trains due to the generation of an *incorrect movement request* by the train.

D2: Possible collision of a train with an object due to the generation of an *incorrect movement request* by the train.

E1: Possible collision of two trains due to the *incorrect transmission of a message from train to TCC* by the mobile communication system.

TABLE 4.1: Hazard table for SatZB

Resource	Functionality							Accident characteristics			
	Location determination	Speed supervision & control	Telegram generation		Message transmission		Telegram generation	Collision			
	Localisation	Command execution	Location report generation	Movement request generation	Train to TCC	TCC to train	MA (Movement Authority) generation	Real-end collision	Head-on collision	Collision of a train with an object on the track	Collision of a train with an object on the level crossing
Train	False							A1		A2	
		False						B1		B2	
			False					C1		C2	
				False				D1		D2	
Mobile comm.					False			E1		E2	
						False		F1		F2	
TCC							False	G1		G2	

E2: Possible collision of a train with an object due to the *incorrect transmission of a message from train to TCC* by the mobile communication system.

F1: Possible collision of two trains due to the *incorrect transmission of a message from TCC to train* by the mobile communication system.

F2: Possible collision of a train with an object due to the *incorrect transmission of a message from TCC to train* by the mobile communication system.

G1: Possible collision of two trains due to the generation of an *incorrect MA* by the TCC.

G2: Possible collision of a train with an object due to the generation of an *incorrect MA* by the TCC.

4.1.3 Hazardous Conditions of SatZB Model

In [102], the concept of generic safety implementation is discussed for the purpose of minimizing the probability and severity of damage via applying safety measures. One of the promising implementation approaches associates with the control of the safety strategy which could have different characteristics, e.g., hazard avoidance, hazard prevention and effect/-consequence minimization. According to Figure 4.1, hazard avoidance is implemented by presenting evidences that all the hazardous conditions are eliminated or there are no hazardous conditions at all, and hazard prevention is realised by the protection measures, e.g., safety functions. Therefore, the hazardous conditions of SatZB model is identified in this subsection. Assuming that the message transmission by the communication system is fully reliable, the hazardous conditions of SatZB model considering the functions presented in Table 4.1 can be shown in Table 4.2. Beside algorithm errors and Petri net structure errors, there might be other hazardous conditions such as configuration errors. Further analysis would be conducted along with the refinement of the system model.

4.2 Function Identification and Allocation

The main task of a train control system is to avoid accidents and hazardous situations. Therefore, all train control systems, including satellite-based train control system such as the SatZB system, have some similar (generic) functions, e.g., assurance of the safety of the routes, supervision and controlling of the train speed. In general, theses functions can be categorised

TABLE 4.2: Hazardous conditions of SatZB model

False function	Possible causes (hazardous conditions)		
	On-board module	TCC module	Localisation unit
Localisation			Algorithm errors, Petri net structure errors, etc.
Command execution	Algorithm errors, Petri net structure errors, etc.		
Location report generation	Algorithm errors, scenario net structure errors, etc.		
Movement request generation	Algorithm errors, scenario net structure errors, etc.		
MA generation		Algorithm errors, scenario net structure errors, etc.	

into two classes: functions for safety and functions for normal operation. Note that we assume the communication system is ideally dependable and safe, therefore it will not be investigated in this work.

4.2.1 Functions for Safety and Their Allocations

As depicted in Figure 4.1, by implementing safety functions (protection measures), the system can be transformed from hazardous states into safe states. In other words, safety functions can prevent accidents from happening when the system is in hazardous states. Safety functions, in general, are defined according to the safety requirements of the system. Based on Figure 4.2, following safety requirements of train control systems could be specified:

- A defined block section can only be reserved by one train at the same time and every invasion of other trains, persons or objects must be detected.
- The train must be prevented from exceeding its permitted speed. Otherwise, a service brake or emergency brake should be issued.
- The train must be prevented form exceeding its permitted moving distance. Otherwise, a service brake or emergency brake should be issued.

With the safety requirements, functions of a system for safety purpose can be identified. In the specific system model of SatZB, the functions and related reactions for safety need to be allocated to each subsystem models, which is shown in Table 4.3. As the SatZB model presented in this thesis is a relatively high level of abstraction model focusing on operation scenarios, the allocation of functions is only deep to the level of scenario nets.

4.2.2 Functions for Normal Operations and Their Allocations

A train control system on the operative level (other three higher levels in the top-down orientation are: strategic level, dispositive level and tactic level [88]) has four generic functions: route control, train control, localisation and data collection. In SatZB, these functions are implemented by the functions listed in the column `Concrete in SatZB` of Table 4.4 based on the functional requirements specification. Table 4.4 also shows the allocation of these functions to the subsystem models of SatZB. Different from Figure 4.3, function identification in this subsection is not based on the system requirements specification, but the functional requirements specification.

4.3 Verification Tasks for the On-board Module of SatZB Model

Based on the hazard analysis, the identification of functions for safety and normal operation as well as their allocations in system model, the verification for the system model should be carried out to demonstrate that no identified hazardous conditions are contained and identified functions for safety and normal operation are functioning as expected. In particular, the verification tasks for the on-board module are specified as follows:

Hazardous conditions:

- The on-board module contains algorithm errors.
- The scenario nets of the on-board module contain structure errors.

Safety functions:

- The on-board module can switch its activated scenario net to the net `OB_SN_Emergency_Stop` from any other nets (except for the nets `OB_SN_Initialisation`, `OB_SN_Registration` and `OB_SN_Logout`).

TABLE 4.3: Functions for safety and their allocations in SatZB model (Note: (1) X(I) means if necessary; (2) the prefix "OB_SN_" of the name of a scenario net is omitted)

Accident characteristics	Speed supervision & control		SatZB model		
	Emergency brake	Speed reduction	On-board module	TCC module	Localisation unit
Rear-end collision	X		Switch to (activates) the scenario net `Emergency_Stop` and issue an emergency brake	Issue an emergency stop command	
Head-on collision	X(I)	X	Switch to the scenario net `Ban_Of_Entry` or `Emergency_Stop` and issue an emergency brake	Issue a command of ban of entry or emergency stop	
Collision of a train with an object on the track	X		Switch to the scenario net `Emergency_Stop` and issue an emergency brake	Issue an emergency stop command	
Collision of a train with an object on the level crossing	X		Switch to the scenario net `Emergency_Stop` and issue an emergency brake	Issue an emergency stop command	
Fire/ electricity problem	X(I)	X	Switch to the scenario net `Conditional_Running` or `Emergency_Stop` and issue an emergency brake	Issue a command of conditional running or emergency stop	
Flood/ avalanches/ landslide	X(I)	X	Switch to the scenario net `Conditional_Running` or `Emergency_Stop` and issue an emergency brake	Issue a command of conditional running or emergency stop	

TABLE 4.4: Functions for normal operation and their allocations in SatZB model (Note: the prefix "OB_SN_" of the name of a scenario net is omitted)

Function	Concrete in SatZB	SatZB model		
		On-board module	TCC module	Localisation unit
Route control (MA generation)	Route establishment		Assign block sections from "free" to "reserved"	
	Route maintenance		Assign block sections from "free" to "reserved", "reserved" to "occupied", and "occupied" to "free"	
	Movement authority issuing		Send the message "mt1" to the train	
	Level crossing protection			
Train control (Command execution)	Expected speed calculation			
	Service braking curve calculation and service brake issuing	Switch the activated scenario net from `Running` to `Ban_Of_Entry`		
	Emergency braking curve calculation and emergency brake issuing	Switch the activated scenario net to `Emergency_Stop`		
	Standby state supervision	Scenario nets `Initialisation` and `Registration`		
	Running state supervision	Scenario net `Running`		
	Conditional running state supervision	Scenario net `Conditional_Running`		
	Ban of entry state supervision	Scenario net `Ban_Of_Entry`		
	Emergency stop state supervision	Scenario net `Emergency_Stop`		
Localisation	Determination of the train location			Location generation
	Calculation of the train speed			
	Supervision of the train integrity			
Data collection	JRU (Juridical Recorder Unit)	Places representing the MMI and the error recorder	Places representing the MMI and the error recorder	

- The on-board module can switch its activated scenario net to the net `OB_SN_Ban_Of_Entry` from other nets (i.e., `OB_SN_Running` and `OB_SN_Conditional_Running`).
- The on-board module can switch its activated scenario net to the net `OB_SN_Conditional_Running` from other nets (i.e., `OB_SN_Running`).

Normal operation functions:

- The on-board module has the possibility to activate every scenario net at least one time.
- The on-board module switches its activated scenario net from one to another as Figure 3.7 depicted.
- The on-board module can not activate two or more than two scenario nets at the same time.

4.4 Summary

This chapter identifies the verification tasks for the on-board module of the SatZB model with respect to system functionalities. These verification tasks will guide the verification work of chapter 6, 7 and 8. The verification tasks specified in this chapter are distinguished into three aspects: hazardous conditions, safety functions and normal operation functions.

Chapter 5

Quality Assurance by Petri Net Analysing Techniques

The employment of the BASYSNET method for developing the SatZB system makes it possible to ensure the quality of the system model by simulation, testing and formal analysis. In this chapter, the quality assurance by Petri net analysing techniques in general is discussed. First, the behavioural properties of Petri nets are formally introduced and represented by an attribute hierarchy model. The interpretation of Petri net behavioural properties as system behaviours are also presented. Second, similar to the introduction of the behavioural properties, the structural properties of Petri nets are introduced and interpreted as system behaviours. Last, coverage-based test generation techniques are discussed. To satisfy the test coverage criterion of all state sequences, one of the many introduced test coverage criteria, reachability tree based and unfolding based methods are proposed to generate the test suites.

5.1 Behavioural Analysis

To analyse a Petri net model, two types of properties can be investigated: those which depend on the initial marking, and those which depend on the topological structure of the Petri net. The former type of properties is referred to as marking-dependent or behavioural properties, whereas the latter one is called structural properties [46]. In this section, we discuss the behaviour properties and the structural properties will be discussed in next section. Nevertheless, both behavioural properties and structural properties will be illustrated by an attribute hierarchy formation described with UML-based notation [103], [104] for a better comprehension of these properties.

The concept model of the attribute hierarchy consists of four levels of abstraction and uses different hierarchical attributes at different levels of abstraction as following:

Property. *Properties* result from the abstraction of characteristics, or rather unite a number of subordinate characteristics. Properties are in general not directly or objectively measurable (e.g. "beauty"). In the sense of UML, it deals with abstract concepts (e.g. classes) which can not be directly instantiated.

Characteristic. *Characteristics* are objectively quantifiable. They are essential for the identification and description of objects. Properties are characterised by characteristics.

Quantity. *Quantities* are special cases of characteristic instantiations. The determination of quantity is restrict to relation scaled characteristics so that there are no ordinal quantities, but ordinal characteristics.

Value. *Values* are the results of measurements, estimations or calculations. A numerical value quantifies the quantity by numerical values.

5.1.1 Behavioural Properties

Boundedness. A P/T system (N, M_0) is *k-bounded* or simply *bounded* if the number of tokens in each place does not exceed a finite number k for any marking reachable from M_0, i.e., $\forall p \in P, M \in [M_0\rangle, M(p) \leq k, k \in \mathbb{N}$. A P/T system (N, M_0) is said to be *safe* if it is 1-bounded. A place is said to be bounded if the maximum number of tokens a place may contain is finite. A P/T system (N, M_0) is bounded if each place is bounded.

Liveness. A P/T system (N, M_0) is *(strongly) live* (or equivalently M_0 is a *live marking of* N) if no matter what marking has been reached from M_0, it is possible to ultimately fire any transition of the net by running some further firing sequences, i.e., $\forall t \in T, M \in [M_0\rangle, \exists M' \in [M\rangle : M'[t\rangle$. A P/T system (N, M_0) is *weakly live (deadlock-free)* if no matter what marking has been reached from M_0, it is possible to ultimately fire at least one transition of the net by running some further firing sequences, i.e., $\forall M \in [M_0\rangle, \exists t \in T : M[t\rangle$. A transition is live if it is potentially firable in all reachable markings. A P/T system (N, M_0) is live if all transitions are live.

Reversibility and Home State. A P/T system (N, M_0) is *reversible* if, for each marking M reachable from M_0, M_0 is reachable from M, i.e., $\forall M \in [M_0\rangle, M_0 \in [M\rangle$. In other words, a reversible P/T system can always go back to the initial marking or state. However, in many cases, the system is not necessary to go back to the initial state but to some (home) state. A

marking M' is a *home state* if, for each marking M reachable from M_0, M' is reachable from M, i.e., $\forall M \in [M_0\rangle, M' \in [M\rangle$.

Reachability. A marking M is *reachable* from a marking M_0 if there exists a firing sequence σ such that $M_0[\sigma\rangle M$. The *reachability graph* $RG((N, M_0)) = (V, E)$ of a P/T system (N, M_0) is a directed graph, where $V = [M_0\rangle$ is the set of vertexes, and $E = \{(M, t, M')|M[t\rangle M'\}$ is set of labelled edges. The labels (M, t, M') are mostly written as t by the labelling $L(E) = \{t \in T|\exists(M, t, M') \in E\}$.

Figure 5.1 is the concept model of Petri nets in the behavioural properties perspective. It is observed that a Petri net system has varies structural properties, e.g., boundedness, liveness, reversibility, home state and reachability. Each of the properties has specific characteristics that are quantified with finite numerical values.

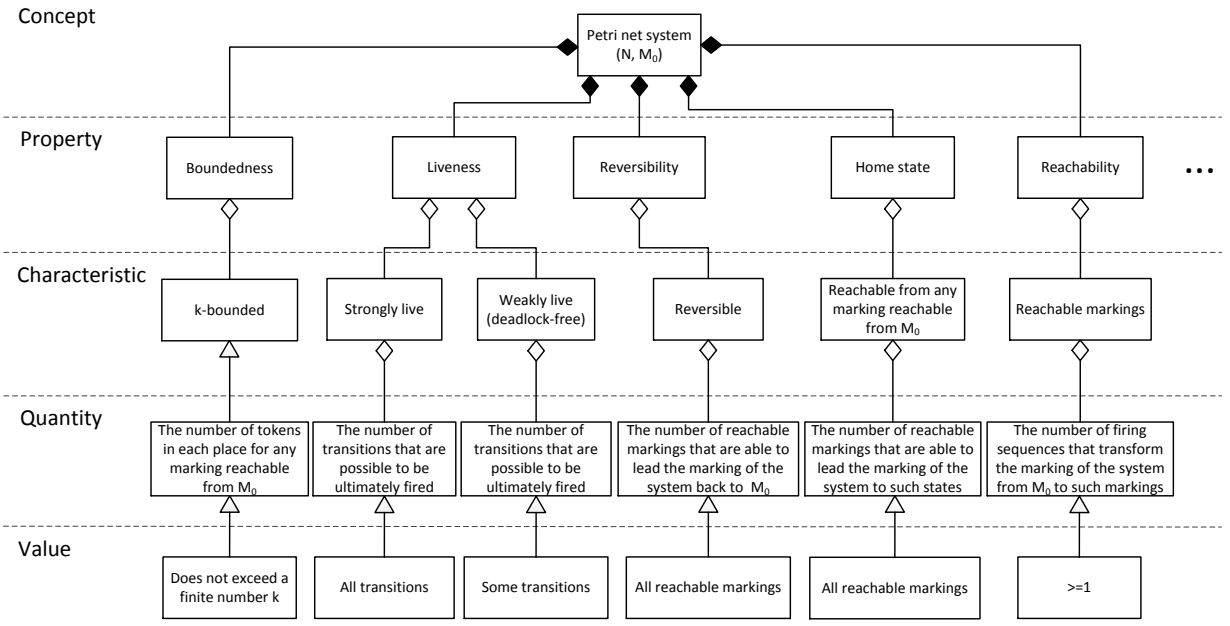


FIGURE 5.1: Concept model of Petri net systems in the behavioural properties perspective

5.1.2 Interpretation of Petri Net Behavioural Properties as System Behaviours

In practice, a Petri net model represents its underlying system which is a real world system. Thus, it is necessary to interpret the behavioural properties of the Petri net model into the behaviours of the underlying system. The interpreting bridges the understanding of the system between the engineers and the customers. Table 5.1 shows the interpretation of Petri net behavioural properties as system behaviours.

TABLE 5.1: Interpretation of Petri net behavioural properties

PN behavioural properties	System behaviours
Boundedness	Whether there are no overflows in the system with respect to a certain initial state
Liveness	Whether the system can always progress with respect to a certain initial state
Reversibility	Whether the system can go back to its initial state at any time
Home states	The system can always turn into these states starting from its initial state
Reachability	Whether the system can turn into certain states starting from its initial state

5.2 Structural Analysis

In this section, we discuss the structural properties of Petri nets. These properties can often be characterised in terms of the incidence matrix of the Petri net and its associated homogeneous equations or inequalities [46]. This mathematical foundation allows one to verify the *absence of errors* in the model by checking some structural properties of the underlying system.

5.2.1 Structural Properties of Petri Nets

Structurally Boundedness. A Petri net N is *structurally bounded* if it is bounded for any finite initial marking M_0, i.e., $\forall M_0 \in \mathbb{N}^{|P|}, M \in [M_0\rangle : M$ is bounded.

Structurally Liveness. A Petri net N is *structurally live* if there exists a live initial marking M_0 for N, i.e., $\exists M_0 \in \mathbb{N}^{|P|}, \forall t \in T, M \in [M_0\rangle, \exists M' \in [M\rangle : M'[t\rangle$.

Conservativeness. A Petri net N is *(partially) conservative* if there exists a positive integer for every (some) place such that the weighted sum of tokens is a constant for every marking M reachable from the initial marking M_0 and for any fixed initial marking M_0, i.e., $\exists y \in \mathbb{N}^{|P|} : \sum M(p_k) \cdot y(k) = \sum M_0(p_k) \cdot y(k) = CONT$ where y is a vector, $k \in [1, |P|]$ and $CONT$ represents a constant.

Repetitiveness. A Petri net N is *(partially) repetitive* if there exists a marking M_0 and a firing sequence σ from M_0 such that every (some) transition occurs infinitely often in σ.

Consistency. A Petri net N is *(partially) consistent* if there exists a marking M_0 and a firing sequence σ from M_0 back to M_0 such that every (some) transition occurs at least once in σ.

Controllability. A Petri net N is *completely controllable* if any marking is reachable from any other marking [46]. In addition, we say a Petri net N is *partially controllable* if there exists a marking that is reachable from any other marking.

As in subchapter 4.1.1, the concept model of Petri nets in the structural properties perspective could be illustrated in Figure 5.2. It is shown that Petri nets have a variety of structural properties, e.g., structurally boundedness, structurally liveness, conservativeness, repetitiveness, consistency and controllability. Each of the properties depends on specific characteristics some of which are quantified by either vectors or numerical values.

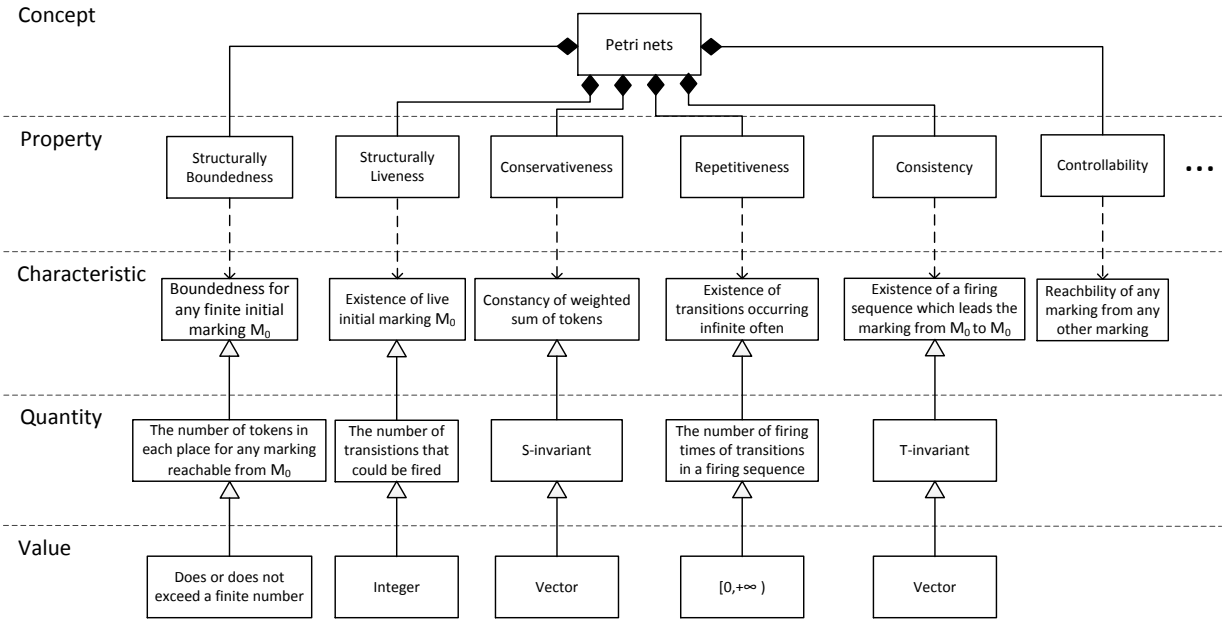


FIGURE 5.2: Concept model of Petri nets in the structural property aspect

5.2.2 Interpretation of Petri Net Structural Properties as System Behaviours

Not only the behavioural properties, but also the structural properties of a Petri net model need to be interpreted into the system behaviours of its underlying system in practice. Table 6.2 shows the interpretation of the structural properties of Petri nets as the behaviours of real systems.

TABLE 5.2: Interpretation of Petri net structural properties

PN structural properties	System behaviours
Structurally Boundedness	Whether there are no overflows in the system
Structurally Liveness	Whether the system can always progress
Conservativeness	Whether the number of items which could be stored/processed/transferred is constant
Repetitiveness	Whether there are processing loops in the system
Consistency	Whether the system can go back to its certain initial states
Controllability	Whether the system will always reach certain states

5.3 Coverage-based Test Generation

Petri nets as formal means of descriptions are usually used as formal specifications of (concurrent) systems, which can be seen as test models in testing the systems. Test cases are derived from the Petri nets and implemented on the systems. On the other hand, Petri nets can be considered as executable models (design models) of (concurrent) systems and transformed into source codes of the system software. In this case, the Petri nets are treated as test objects (SUT, system under test). Generally speaking, testing methods can be classified into *program-based*, which select test cases according to the information contained in the program, and *specification-based*, which derive test cases from the requirements specification [105]. When Petri nets are used as test models, formal specifications of systems, the Petri net testing methods belong to specification-based testing. If Petri nets are considered as executable models of systems, they can also be tested against other specifications. In this sense, the Petri net testing methods have the features of program-based testing. In testing, no matter what kinds of testing, program-based or specification-based, a *well-defined testing method* should be defined. In [105] and [106], the authors argued that a well-defined testing method should contain at least two components, a criterion for selecting test cases and determining when the testing can stop, and a method of observing the dynamic behaviour of a system during the test execution. Considering this requirement, [105] defines three test selection criteria, i.e. transition coverage, state coverage and flow coverage, concerning with the structure of PrT nets (predicate transition nets [107], [108], high-level Petri nets) which are taken as the test objects by the authors. These criteria are used as adequacy criteria to analyse the test adequacy of the testing. In [109], three coverage criteria, i.e. transition coverage, place coverage and marking coverage, are also defined to select test cases. In contrast to [105], [109] applies these criteria to the test model (LPrTPN, Labeled Prioritized Time Petri Nets) of the SUT for test generation.

In [110] and [111], three *entity-based* classes of CPN coverage criteria, i.e. transition-based coverage criteria, event-based coverage criteria and state-based coverage criteria, are proposed to derive test cases for testing cooperating robotic systems. In addition, the transition-based coverage criteria are distinguished between *all transitions, all transition pairs* and *all transition sequences*. The even-based coverage criteria take *all events, all event pairs* and *all event sequences* into account. The state-based coverage criteria cover *all states, all state pairs* and *all state sequences*. In [112], the criteria of reachability graph coverage, transition coverage, state coverage, depth coverage and goal coverage are introduced to generate test sequences from PrT nets in the tool ISTA (Integration and System Test Automation). However, these works only succeeded to bring up coverage criteria for selecting test cases, either for the coverage of the Petri nets of SUT or for the coverage of the Petri nets of test models. They haven't proposed techniques on how to generate test cases satisfying the criteria, and to our knowledge, few works have tackled this issue.

In [113], a technique called *cause-effect-net-concept*, derived from a program code testing concept *cause-effect graphing*, is proposed to generate test cases for the simulation of high-level Petri nets (PrT nets) in a systematic way. The test cases are derived from process nets, which are defined as: a process net describes one single behaviour of a system or of a Petri net. In [114], two CPN based test suite generation methods for conformance tests, Coloured Petri net Tree (CPT) method and Coloured Petri net Graph (CPG) method, are proposed. The CPT method makes use of the CP-trees (reachability trees) of CP-nets which are reduced by equivalent markings introduced in [115]. The CPG method regards the CP-graphs (reachability graphs) as finite state machines (FSM) and test suites are generated by applying existing methods based on FSM. The CP-graphs in this method are constructed directly from the CP-trees that have been reduced by equivalent markings. Recently, Krause [116], [41] has proposed a test generation technique based on the unfoldings [117], [118], [119] of SPENAT (Safe Place Transition Nets with Attributes) specifications. Nevertheless, these works failed to discuss the Petri net based test coverage criteria in a systematic manner.

In this section, we introduce some coverage criteria for selecting test cases as well as test generation techniques for the strongest coverage criterion of all state sequences. Note that the coverage criteria discussed here refer to the coverage adequacy of test models of the SUT.

5.3.1 Test Coverage Criteria

A Petri net based test case is defined as a pair consisting of an initial state and a finite firing sequence, the term *test sequence* could be used to describe such a test case. Normally, with a

common initial state, a Petri net will generate a set of test sequences. We call this set of test sequences a *test suite*.

Transition-based coverage criteria. Following transition-based coverage criteria may be considered:

- *All transitions*: each transition in the Petri net (test model) is covered by at least one test sequence;
- *All transition sequences*: each possible transition sequence is covered by at least one test sequence.

Transition-instance-based coverage criteria (High-level Petri Nets). Following transition-instance-based coverage criteria may be considered:

- *All transition instances*: each transition and transition instance (binding element) is covered by at least one test sequence;
- *All transition instances sequences*: each possible sequence of transitions and transition instances is covered by at least one test sequence.

State-based coverage criteria. Following state-based coverage criteria may be considered:

- *All states*: each state (marking) reachable from the initial state is covered by at least one test sequence;
- *All state sequences*: each possible sequence of states is covered by at least one test sequence.

Interested-state-based coverage criteria. Following interested-state-based coverage criteria may be considered:

- *All interested states*: each interested state (marking) reachable from the initial state is covered by at least one test sequence;
- *All interested state sequences*: each possible sequence of interested states is covered by at least one test sequence.

In order to illustrate the subsumption relations of the presented coverage criteria, the subsumption hierarchy of the coverage criteria is shown in Figure 5.3. It is easy to observed that the criterion of all state sequences subsumes the criteria all states and all interested state sequences. In fact, a transition instance is mapped to an arc in the reachability graph, so the criterion all state sequences also subsumes the criterion all transition instance sequences. Since a transition in a high-level Petri net (e.g. CPN) could have different instances according to different token colours, the criterion all transition instance sequences subsumes the criterion of all transition sequences and the criterion of all transition instances subsumes the criterion of all transitions. The interested states are parts of the state space, hence the criterion of all states subsumes the criterion of all interested states.

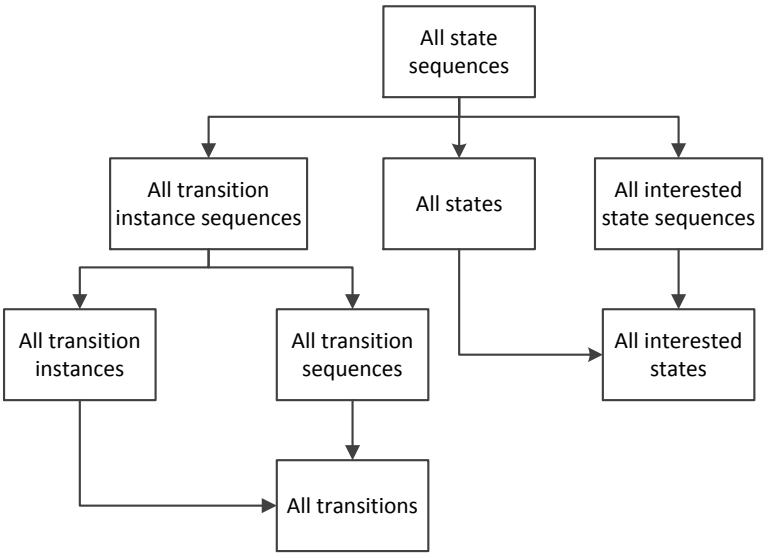


FIGURE 5.3: Subsumption hierarchy of coverage criteria

5.3.2 Modelling the System and Its Environment

Test generation based on Petri nets needs to model the intended behaviour of the SUT first. To imitate the operation of the system in its real operating environment, it is necessary to establish a model of the environment. This environment model encodes the environment assumptions for the purpose of verifying and validating the system by least effort. Otherwise the environment model is a model that allows all possible interaction sequences.

5.3.3 Test Suite Generation

Once a test model is developed with Petri nets, a test suit could be generated with respect to a specific initial marking. In order to generate test suites satisfying the strongest coverage criterion of *all state sequences*, *reachability tree* (RT) based and *unfolding* based methods are proposed in this section.

5.3.3.1 RT-based Method

In general, the state space of a Petri net could be represented by the reachability graph of the net. Intuitively, a test sequence can be derived from a path of the graph starting from the initial marking. A test suite consists of all the test sequences derived from all the paths of the graph. Nevertheless, this is only practicable for the nets that the reachability graphs of which are relatively small and straightforward. If the reachability graph of a Petri net is rather big and sophisticated, e.g., it contains many circles and branching paths, deriving test sequences from the reachability graph is very difficult. For this reason, the RT-based method could be adopted.

A *reachability tree* consists of a tree whose nodes represent markings of the Petri net and whose arcs represent the possible changes in state resulting from the firing of transitions. In fact, a reachability tree is equivalent to the reachability graph of the Petri net in the sense that it contains all reachable markings and all possible firing sequences. A reachability tree will be infinite if there exists circles in the reachability graph or if the state space is infinite. Therefore, it is necessary to reduce it to a finite size for practical uses.

Reachability Tree of Low-level Petri Nets. For low-level Petri nets, the reduction of reachability tree could be done by using a special symbol ω for the number of tokens in the places which could have arbitrarily large number of tokens [120]. ω satisfies $\omega + \alpha = \omega$, $\omega - \alpha = \omega$ and $\alpha < \omega$ for any natural number α. In the reachability tree, each node is labelled with a marking and each arc is labelled with a transition. The initial node (root of the reachability tree) is labelled with the initial marking. Inspired by [120], a *finite reachability tree* can be constructed as follows:

1. Given a node x in the tree, "new" nodes are added to the tree for all markings that are directly reachable from the marking of the node x. The arc from the node x to each "new" node is labelled with the transition, by firing which the marking is transformed from the marking of the node x to the marking of the "new" node.

2. Repeat the process of step 1.

2.1. If the marking of a "new" node is equal to the marking of an existing node on the path from the root node to the "new" node, the "new" (duplicate) node becomes a terminal (dead) node.

2.2. If the marking of a "new" node x is grater than the marking of a node y on the path from the root node to the node x, then those components (number of tokens in a corresponding place) of the marking of the node x which are strictly greater than the corresponding components of the marking of the node y are replaced by the symbol ω. In general, we say a marking M_1 of a Petri net is greater that another marking M_2 of the net if at least one component of marking in M_1 is greater than it is in M_2 and the each of the rest components of marking in M_1 is no less than it is in M_2. For example, M_1 and M_2 are two markings of the Petri net $N = (P, T, F)$ where $P = (p_1, p_2, p_3)$. If $M_2(p_1) = 1 > M_1(p_1) = 0$ and $M_2(p_2) = M_1(p_2) = 1$ and $M_2(p_3) = M_1(p_3) = 0$, then the marking M_2 is greater than M_1. In this case, the first component of the marking M_2 is replaced by ω and thus $M_2 = (\omega, 1, 0)$.

The reason for reducing the reachability tree by step 2.1 is because all markings reachable from this node have already been added to the tree by the "old" node with the identical marking. For step 2.2, since the marking of node x is greater than the marking of node y, any possible firing sequences from node y is also possible from node x. In particular, the firing sequence that transforms the marking from node y into node x can be repeated infinitely, and each time increasing the number of tokens in those places corresponding to the ω components. Therefore, the number of tokens in these places can become arbitrarily large.

Apart from using the symbol ω, [121] presents a *modified reachability tree* for Petri nets following the suggestion of [122] to use the expression $a + bn_i$ rather than ω to represent the value of the components of a marking. More details can be found in [121].

Reachability Tree of High-level Petri Nets. For high-level Petri nets, the reachability tree can be reduced by *equivalent markings* [123] which is a generalization of duplicate markings. In order to have an easy entry for the readers, [123] and [115] used the "five dining philosophers" example to introduce the notion of equivalent markings. For example, the marking that represents philosopher number one is eating with the both folks next to him (left and right hand side folks) is said to be equivalent to the marking that represents philosopher number two is eating with the both folks next to him, but not equivalent to the marking that represents philosopher number two is eating with the folks that not both are next to him. The explanation is that all philosophers should "behave in the same way" ignoring the identity of eating philosophers. For the reduced reachability tree by equivalent markings, only one element of each class of equivalent markings is developed further, and when a marking has several direct successors which are equivalent, only one of them is included in the tree. [123] pointed

out that the relation of equivalent markings is determined by the persons who analyse the system, and it must respect the inherent nature of the system. Formal definitions of reachability tree for high-level Petri nets and for CP-nets as well as the algorithms of producing the reachability tree can be seen in [123] and [115], respectively. Moreover, the reachability tree of high-level Petri nets can be further reduced with the reduction methods presented previously for the low-level Petri nets.

Test Suite Generation.

1. Generate the finite reachability tree of the Petri net.
2. Identify the initial node and leaf nodes (dead nodes) of the tree. Each path from the initial node to a leaf node is a test sequence and all the test sequences comprise the test suite.

5.3.3.2 Unfolding-based Method

The RT-based method, in essence, makes use of reachability graphs which are unfolded into reachability trees representing all the reachable markings and firing sequences of Petri nets. This makes it possible to alleviate the *state explosion problem* in analysing Petri nets with their state sates (reachability graphs) because finite reachability trees are possible with appropriate reduction methods. To avoid the state explosion problem, another technique using the notion of *unfolding* was proposed by McMillan [117], [118]. The unfolding of a Petri net is a *partially ordered net* [119] containing the information of all the reachable markings and preserving the information of concurrency of the original Petri net. It is a *labelled net* [124] whose places are labelled with places of the original Petri net and whose transitions are labelled with transitions of the original Petri net. The unfolding of a Petri net is usually an infinite net, which give rise to problems of analysing the Petri net. For this reason, McMillan also proposed an algorithm to construct a *truncated unfolding* (called *finite complete prefix* in most of the later literature, e.g. [119], [40] and [125]), a finite fragment of the unfolding, which is sufficient to represent all the reachable markings. The formal introduction of unfoldings and the algorithm of constructing the finite complete prefixes of unfoldings will be presented in chapter 7.

Test Suite Generation.

1. Construct the finite complete prefix of the unfolding of the Petri net.
2. Identify all the *local configurations* (see chapter 7) of the events of the finite complete prefix which have no successor events. Every *linearisation* (see chapter 7) of a local

configuration corresponds to a test sequence. All the test sequences comprise the test suite.

5.4 Summary

This chapter discusses the quality assurance by Petri net analysing techniques. The behavioural analysis techniques and structural analysis techniques rely on the behavioural properties and structural properties of Petri nets and their interpretations as system behaviours, respectively. The behavioural properties and structural properties of Petri nets are illustrated by attribute hierarchy models for a better comprehension. Model-based testing with Petri nets needs to specify the test coverage criteria and develop the test generation methods. For this purpose, four classes of test coverage criteria have been discussed on one hand: transition-based coverage criteria, transition-instance-based coverage criteria, state-based coverage criteria and interested-state-based coverage criteria. On the other hand, two test suite generation methods satisfying the coverage criterion of all state sequences have been presented: reachability tree based and unfolding based method.

Notice that this chapter discusses the Petri net analysing techniques in a general way. When these techniques are applied to a specific system, specific refinements for each technique are required. Examples of applying these techniques to the SatZB model will be presented in chapter 6, 7 and 8.

Chapter 6

Structural Verification Using Open Nets

Once we have established a Petri net model for system development, the properties of the net that are independent of the initial marking, but depend on its topological structure need to be investigated to make sure that no undesired properties, e.g., traps and co-traps are included in the model. These types of properties are structural properties of Petri nets. Prior to verify the Petri net model against its underlying system's requirements specification, the structure errors should be eliminated. This is done by structural verification of the net.

In this chapter, first open nets are formally defined and several application contexts (environments) are specified. Second, the reproducibility of empty markings of open nets is discussed based on Lautenbach's theory. Third, the identification of dead transitions in open nets is presented. Fourth, open nets' ability of terminating in empty markings is investigated. Finally, an example of verifying the structural properties of consistency and controllability for the generic scenario net of the on-board module of SatZB model is provided to illustrate the presented theory.

6.1 Open Nets for Structural Analysis

In this work, our solution for modelling large-scale systems is the approach of modularisation which decomposes the system model into smaller modules (component models) and these modules are interacted via interfaces. This is realised in chapter 3 by hierarchically model (top-down approach) the system with CP-nets. Each of these modules, in fact, is an *open system* which interacts with the surrounding environment (e.g., other modules). To analyse such a large-scale system model that consists of smaller modules, in particular, if the system

model is developed by different persons or work groups, it is necessary to analyse each module independently by the respective developers. For this purpose, the notion of *open nets* are suitable to describe the modules of a system model.

According to the survey of literatures (e.g., [37], [126], [127], [128]), most applications of the notion of open nets are dedicated to compose systems or process models (bottom-up approach). In [37], open high-level Petri nets are defined to model the scenarios of a railway level crossing control system, based on which the cooperability of system components is ensured by adequate integration and composition techniques for open nets. In [126], the author exploited open Petri nets for the modelling of workflows across organisational boundaries. In [127], open Petri nets are introduced in order to model the behaviours of open concurrent systems by means of Petri nets. In [128], a framework based on open Petri nets is proposed for the specification of behaviour-preserving reconfigurations of systems modelled as Petri nets. In contrast to the presented works, we use open nets to decompose the system model and focus on the structural analysis of the open nets.

6.1.1 Open Nets

An *open net* is a Petri net (ordinary P/T net) with a distinguished set of *open places*, through which the net interacts with the surrounding environment. As a consequence, tokens can freely appear or disappear on open places. An open place can be either an *input (boundary) place* where the environment can put tokens, or an *output (boundary) place* whose tokens can be removed by the environment.

Definition 6.1 (Open nets)**.** Let $N = (P, T, F)$ be a Petri net. A place p is an *input (output) boundary place* iff ${}^\bullet p = \emptyset (p^\bullet = \emptyset)$. A transition t is an *input (output) boundary transition* iff ${}^\bullet t = \emptyset (t^\bullet = \emptyset)$. An *open net* is a Petri net defined as a 5-tuple $N_O = (P_I, P_O, P, T, F)$, where (P, T, F) is a Petri net, P_I is the set of input boundary places and $P_I \neq \emptyset$, P_O is the set of output boundary places and $P_O \neq \emptyset$, $P_I \cap P_O = \emptyset, P_I \subseteq P$, $P_O \subseteq P$.

6.1.2 Open Nets in Environmental Contexts

An open net with a fixed structure, interacts with different types of environments might lead to different behaviours including undesired ones. Therefore, when we consider the structural properties of an open net, it is necessary to extend the open net with its application environment. For open nets, different types of environments are distinguished depending on the number of tokens that an environment can produce for each input boundary place and consume from each output boundary place. Table 6.1 specifies three types of environments.

TABLE 6.1: Environment types

Name of the environment	Number of tokens that can be fed to each input boundary place	Number of tokens that can be consumed from each output boundary place
Normal environment	$infinite$	$infinite$
1-configured environment	1	$infinite$
Absorbing environment	0	$infinite$

In the following, we formally define the open nets in different environmental contexts. Note that when we discuss open nets in environmental contexts (definition 6.2, 6.3, 6.4), the necessary conditions of ${}^\bullet P_I = \emptyset$ and $P_O{}^\bullet = \emptyset$ of open nets are assumed to be neglected, which allows the open nets to be extended by adding additional nets to their boundary places. This is valid if the environment nets are added only for analysis purposes and however can exchange information (tokens) with the open nets.

Definition 6.2 (Open nets in normal-environment contexts)**.** Let $N_O = (P_I, P_O, P, T, F)$ be an open net, then we define the *normal-environment* of N_O as $E = T_I \cup T_O$, where $T_I = {}^\bullet P_I, T_O = P_O{}^\bullet, T_I \cap T = \emptyset, T_O \cap T = \emptyset$ and $T_I \cap T_O = \emptyset$. We denote an *open net in normal-environment context* as $N_E = (T_I, T_O, P_I, P_O, F_E, P, T, F)$, where N_E is a Petri net, (P_I, P_O, P, T, F) is an open net, and $F_E \subseteq (T_I \times P_I) \cup (P_O \times T_O) \to \{0, 1\}$.

Definition 6.3 (Open nets in 1-configured-environment contexts)**.** Let $N_E = (T_I, T_O, P_I, P_O, F_E, P, T, F)$ be an open net in normal-environment context, then we define $(N_E, M_{01}) = (P_{E_1}, F_{E_1}, T_I, T_O, P_I, P_O, P, T, F, M_{01})$ as an *open net in 1-configured-environment context*, where $P_{E_1} = {}^\bullet T_I$, $F_{E_1} \subseteq P_{E_1} \times T_I \to \{0, 1\}$, $\forall p \in P_{E_1}, M_{01}(p) = 1$.

Definition 6.4 (Open nets in absorbing-environment contexts)**.** Let $N_O = (P_I, P_O, P, T, F)$ be an open net, then we define the *absorbing-environment* of N_O as $E_O = T_I \cup T_O$, where $T_I = {}^\bullet P_I = \emptyset, T_O = P_O{}^\bullet$ and $T_O \cap T = \emptyset$. We denote an *open net in absorbing-environment context* as $N_{E_O} = (T_O, P_I, P_O, F_{E_O}, P, T, F)$, where N_{E_O} is a Petri net, (P_I, P_O, P, T, F) is an open net, and $F_{E_O} \subseteq (P_O \times T_O) \to \{0, 1\}$.

6.2 Reproducibility of Empty Markings of Open Nets

The reactions (behaviours) of an open net for two identical inputs from its surrounding environment often need to be the same no matter when the inputs appear. This means that at both moments before getting the inputs, the states of the net should be identical. Normally, these two identical states (usually initial states) can be regarded as empty if the components

(places) of the net that are severed as invariable resources are omitted. Consequently, we say that the open net should be capable of reproducing its empty (initial) marking. For instance, for a scenario net of the on-board module, it is required to have the ability to return to its initial state (omit those places that will not affect the data processing, e.g., places used to record specific data) after some specific data processing processes. Ignoring the places used to record data and places served as resources that are connected with double directed (test) arcs in Petri net models, the initial marking of a scenario net is empty. As a consequence, the scenario nets are required to be able to reproduce their empty markings under the assumption given forwards.

Definition 6.5 (Reproducibility [38])**.** Let $N = (P, T, F)$ be a Petri net and M_0 a marking of N. Then a marking M ($M \in [M_0\rangle$) is *reproducible* iff there exists a marking $M' \in [M\rangle$, $M' \neq M$ s.t. $M \in [M'\rangle$.

Theorem 6.6. *Let $N = (P, T, F)$ be a Petri net, j be a non-negative T-invariant of N, then the empty marking $\emptyset$ of N is reproducible by realising $(k \cdot j)$ for $k \in \mathbb{N}$ iff the net representation N_j of j does neither contain a trap nor a co-trap [38].*

Proof. Proof can be seen in [38]. □

6.2.1 Reproducibility of Empty Markings of Open Nets in Normal-environment Contexts

The reproducibility of the empty (initial) marking of an open net N_O in the normal-environment context can be directly investigated by using Theorem 6.6 since N_E is a normal Petri net.

6.2.2 Reproducibility of Empty Markings of Open Nets in 1-configured-environment Contexts

For an open net in the 1-configured-environment context, Theorem 6.6 provides a necessary but not sufficient condition for the reproducibility of the empty marking of the open net, because the entry corresponding to an input boundary place of the support of a non-negative T-invariant could be greater than one. It means that the input boundary place can be marked more than one time, which is in conflict with the specification of open nets in 1-configured-environment contexts.

For example, Figure 6.1 shows an open net N_O where p_1 is an input boundary place and p_3 is an output boundary place. The net N_E (open net in normal-environment context) is

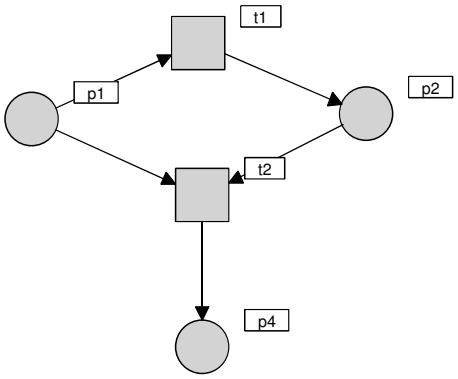


FIGURE 6.1: Open net N_O

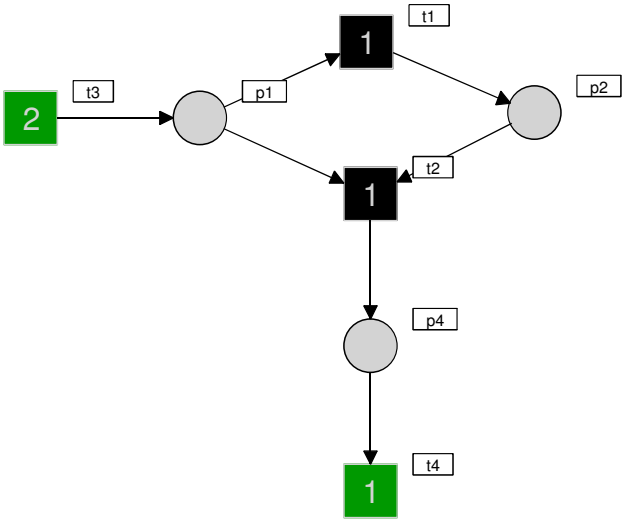


FIGURE 6.2: A T-invariant of N_E

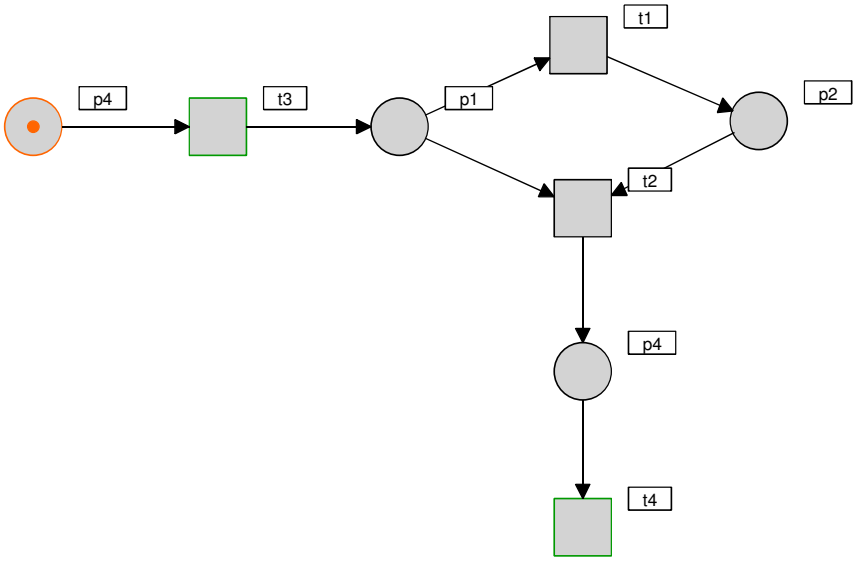


FIGURE 6.3: Open net in 1-configured-environment context (N_E, M_{01})

depicted in Figure 6.2, where t_3 is an input boundary transition and t_4 an output boundary transition. Vector j is a T-invariant, where $j^T = [2\ 1\ 1\ 1]$ shown in Figure 6.2. According to the definition of net representation, N_E is also the net representation N_j of j which contains neither a trap nor a co-trap. Note that the checking of traps and co-traps can be easily carried out by computer-supported tools such as `Poseidon 2.0` [43]. So the empty marking $\emptyset$ of N_O in the normal-environment context is reproducible by realising $k \cdot j$ for $k \in \mathbb{N}$ according to Theorem 6.6. However, the empty marking $\emptyset$ of N_O in the 1-configured-environment context depicted in Figure 6.3 is *unreproducible* even though N_E does not contain a co-trap or a trap. It is easy to find out that t_2 is a dead transition in Figure 6.3.

Proposition 6.7. *Let $N_O = (P_I, P_O, P, T, F)$ be an open net, j be a realisable T-invariant [1] of N_E, where $N_E = (T_I, T_O, P_I, P_O, F_E, P, T, F)$. The empty marking $\emptyset$ of N_O in the 1-configured-environment context is reproducible by realising j iff the net representation N_j of j does neither contain a trap nor a co-trap and $\forall t_i \in T_I, 0 \leq j(t_i) \leq 1$.*

Proof. (Necessity) The value of the $i\,th$ entry of j indicates the firing times of the corresponding transition in the firing sequence that realise j. $\forall t_i \in T_I, 0 \leq j(t_i) \leq 1$ means each input boundary transition can fire at most once. In other words, each input boundary place can be filled with at most one token. According to Theorem 6.6, if the net representation N_j of j does neither contain a trap nor a co-trap, then the empty marking $\emptyset$ of N_O in the 1-configured-environment context is reproducible by realising j.

(Sufficiency) Assuming that the empty marking $\emptyset$ of N_O is reproducible, then there exists a T-invariant j of N_E and the net representation N_j of j does neither contain a trap nor a co-trap according to Theorem 6.6. Since an open net in the 1-configured-environment context implies that each input boundary transition can fire at most once, namely $\forall t_i \in T_I, 0 \leq j(t_i) \leq 1$. □

Remark 6.8. Proposition 6.7 provides a sufficient as well as a necessary condition for reproducing the empty marking of an open net in 1-configured-environment context.

6.3 Dead Transitions in Open Nets

As stated, once a co-trap is unmarked, it remains unmarked. We have an intuitive conclusion that if the initial marking of an open net is empty and there exists a co-trap, then the open net contains dead transitions but still has the possibility to reproduce the empty marking by firing other transitions. Therefore, it is important to make sure that the empty marking of an open net is reproducible as well as no dead transition is contained in the open net.

Definition 6.9 (Dead transition)**.** A transition t is *dead* in (N, M_0) iff $\forall M \in [M_0\rangle : t$ is not enabled.

Theorem 6.10. *Let $N = (P, T, F)$ be a Petri net, there exists a dead transition in $(N, \emptyset)$ iff N contains a co-trap [38].*

Proof. Proof can be seen in [38]. □

[1] A T-invariant of a Petri net might be not realisable, namely the $i\,th$ entry of a T-invariant j does indicate the firing times of the corresponding transition in a firing sequence only if such a firing sequence does exist at all. Examples and related theory can be seen in [38]. Idiotically, one can run the Petri net to verify the realisation of a T-invariant.

6.3.1 Dead Transitions in Open Nets in Normal-environment Contexts

Corollary 6.11. *Let $N_O = (P_I, P_O, P, T, F)$ be an open net, there exists a dead transition t_{dead} in $(N_E, \emptyset)$ such that $N_E = (T_I, T_O, P_I, P_O, F_E, P, T, F)$ and $t_{dead} \in T \wedge t_{dead} \notin (T_I \cup T_O)$ iff N_E contains a co-trap.*

Proof. N_E is a normal Petri net, then the conclusion of Corollary 6.11 is obvious according to Theorem 6.10. □

6.3.2 Dead Transitions in Open Nets in 1-configured-environment Contexts

If the net N_E of an open net in normal-environment context contains a co-trap, then there exists dead transitions in $(N_E, \emptyset)$ according to Corollary 6.11. However, for an open net in the 1-configured-environment context, even if there is no co-trap in N_E, a dead transition could be contained in (N_E, M_{01}).

Definition 6.12 (Path)**.** Let $N = (P, T, F)$ be a Petri net and $w = (u_0, u_1, \cdots, u_n)$ a sequence where $n \geq 1$ and $u_i \in P \cup T$ for $i = 0, 1, \cdots n$. Then w is a *(directed) path* [129] from u_0 to u_n iff $\{(u_0, u_1), (u_1, u_2), \cdots, (u_{n-1}, u_n)\} \subseteq F \wedge u_i \neq u_j$ for $i \neq j$.

Definition 6.13 (Converging Transitions)**.** Let $N_O = (P_I, P_O, P, T, F)$ be an open net. $\forall t \in T$, we define $\uparrow t$ as the *converging transitions of* t, which is a set of transitions of N_O such that $\uparrow t \subseteq T$ and $\forall t'' \in \uparrow t$, t'' lies on at least one (directed) path which is from an input boundary place to the transition t.

Proposition 6.14. *Let $N_O = (P_I, P_O, P, T, F)$ be an open net, $|P_I|$ the amount of input boundary places. $\forall t \in T$, the number of input places of t is $|{}^\bullet t|$. t is a dead transition in (N_E, M_{01}) if $(|P_I| < |{}^\bullet t|) \wedge (\forall t' \in \uparrow t, |{}^\bullet t'| \geq |t'^\bullet|)$.*

Proof. If $\forall t \in T, \forall t' \in \uparrow t, |{}^\bullet t'| \geq |t'^\bullet|$, then the amount of tokens on the net will never increase after firing a transition $t' (t' \in \uparrow t)$, which means that the number of tokens on the net is at most $|P_I|$. If $|P_I| < |{}^\bullet t|$, then there will be at least one input place of t cannot be filled, namely, $\exists p \in {}^\bullet t, M(p) = 0$. Therefore, t is a dead transition. □

Remark 6.15. Proposition 6.14 only provides a sufficient condition for the determination of a dead transition in an open net in the 1-configured-environment context.

6.4 Open Nets' Ability of Terminating in Empty Markings

Following the system requirements, a scenario net is not only required to have the possibility to go back to its empty (initial) state, but also needed to be assured that it will always terminate in an empty state. In this case, the specification of open nets in absorbing-environment contexts is applicable.

Definition 6.16 (Termination)**.** An open net is said to be *terminated* iff no transition is enabled after processing the input tokens produced by the environment.

Proposition 6.17. *Let $N_O = (P_I, P_O, P, T, F)$ be an open net, $N_{E_O} = (T_O, P_I, P_O, F_{E_O}, P, T, F)$ the open net in absorbing-environment context and M_0 a non-empty marking of N_{E_O}. Suppose that an enabled transition in (N_{E_O}, M_0) will fire eventually. Then the net N_{E_O} will terminate in an empty marking (in other words, N_{E_O} will reach an empty marking eventually from a non-empty marking), i.e., $\forall M \in [M_0\rangle, \emptyset \in [M\rangle$ and the empty marking $\emptyset$ is a dead marking iff N_{E_O} contains no trap.*

Proof. (Sufficiency) If the net N_{E_O} can always terminate in the state of empty marking from any non-empty marking, i.e., $\forall M \in [M_0\rangle, \emptyset \in [M\rangle$, then all tokens on the places of the net N_{E_O} can be removed through the output boundary transitions. This, however, is impossible if N_{E_O} contains a trap because once a trap is marked, it remains marked. If the empty marking $\emptyset$ is not dead, then there exists at least one transition $t, t \in T \cup T_O$ is enabled in the empty marking $\emptyset$. This is in contrast to $\forall t \in T \cup T_O, {}^\bullet t \neq \emptyset$. Therefore, the empty marking $\emptyset$ is dead.

(Necessity) Assuming that N_{E_O} contains no trap, then there are no output boundary places because every such place is a minimal trap. Let H, where $H \neq \emptyset$, be the set of marked places of (N_{E_O}, M_0). Then $H^\bullet \neq \emptyset$ since H contains no output boundary places. Let $t \in H^\bullet$, then t is enabled. Suppose that an enabled transition t will fire eventually over time. Then if there exists such a enabled transition t at all, the set of marked places H is still non-empty. In other words, If there is not a enabled transition t any more, then $H = \emptyset$, which indicates that the net N_{E_O} is in an empty marking and no transition is enabled, namely, the empty marking $\emptyset \in [M_0\rangle$ and it is dead. □

6.5 Application Example

In this section, we provide an application example using open net theory to verify some structural properties (i.e., consistency and controllability) of the scenario nets of the on-board

module. First of all, the interpretations of structural properties as the behaviours of the on-board module (or scenario nets) are necessary and presented in Table 6.2. Additionally, the column "Verification tasks" indicates the related verification tasks identified in chapter 4.

TABLE 6.2: Interpretation of structural properties

PN structural properties	Behaviours of On-board module	Verification tasks
Structurally Boundedness	Whether there are no overflows in the model	
Structurally Liveness	Whether there always exists at least one transition in the model that is enabled	
Conservativeness	Whether there is no overflow of information to be stored	
Repetitiveness	Whether there are loops in the process of switching scenario nets	
Consistency	Whether the empty marking of the scenario nets is reproducible	*The scenario nets contain no structure errors*
Controllability	Whether the scenario nets will always terminate in empty marking	*The scenario nets contain no structure errors*

6.5.1 Scenario Nets as the Semantics of Open Nets

Observing the scenario nets of the on-board module presented in chapter 3, we can find that different scenario nets have a similar structure and mechanism which could be abstracted as the generic structure of the scenario nets (see Figure 6.4).

In Figure 6.4, tokens on the places `Message_IN` and `Location_IN` represent the messages sent by the traffic control centre and the localisation unit, respectively. Place `Flag_IN` is the precondition of activating the scenario net. Once the place `Flag_IN` is marked, the scenario net is activated. Otherwise, it is "dead" (the messages on places `Message_IN` and `Location_IN` will/can not be received by firing the transition `Receiving MSG` and `Receiving LOC`). The message received by the place `Flag_IN` is used for determining the next activated scenario net. For example, if the transition `Switch to Other Scenarios` fires, then token on the place `Flag_IN` will be taken and added to the place `Flag_OUT`. The scenario net that takes the place `Flag_OUT` as the precondition of activation will be activated as soon as the place `Flag_OUT` is marked. If the transition `Stay in Current Scenario` fires, then the token remains on the place `Flag_IN`, namely, the current scenario net keeps

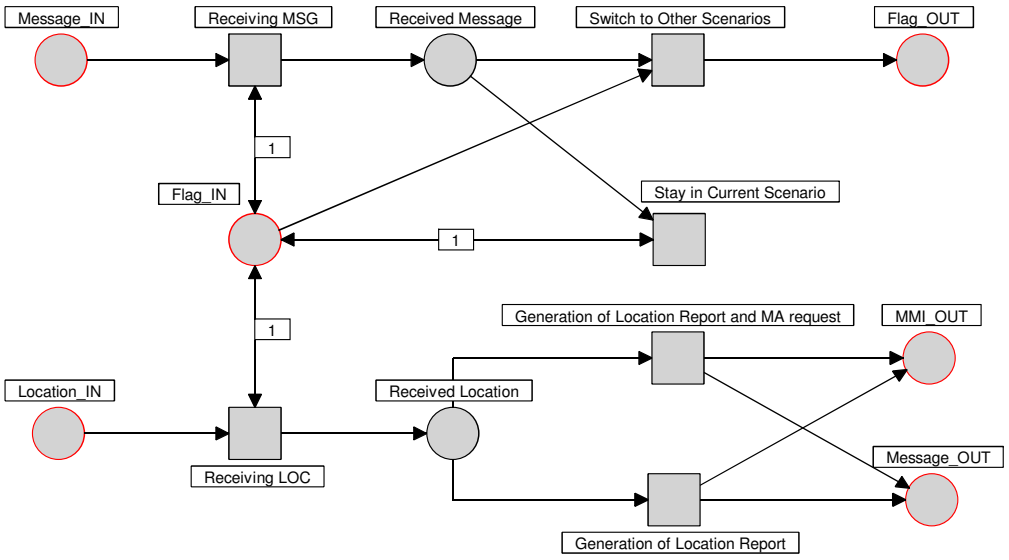


FIGURE 6.4: Open net: the generic structure of scenario nets

in activated state and ready to receive the next messages. The message received by the place `Location_IN` is taken for generating location reports and/or MA requests that are sent out by the place `Message_OUT`. Place `MMI_OUT` indicates the location reports and MA requests to the driver.

According to the definition of open nets, the generic scenario net depicted in Figure 6.4 is an open net in which places `Message_IN`, `Location_IN` and `Flag_IN` are input boundary places and places `Flag_OUT`, `MMI_OUT` and `Message_OUT` are output boundary places.

6.5.2 Consistency: Reproducibility of Empty Markings of Scenario Nets

The net N_E of the scenario net (Figure 6.4) in normal-environment context is shown in Figure 6.5. Transitions `t1,t2,t3,t4,t5` and `t6` represent the environment of the scenario net. To be exact, transitions `t1,t2` and `t3` are input boundary transitions, and transitions `t4,t5` and `t6` are output boundary transitions. An input boundary transition can fire arbitrarily. Using computer-supported tools such as `Poseidon 2.0` [43], we obtain all the T-invariants of the net N_E. One of the T-invariants j_1 (realisable) is shown in Figure 6.5 where the set of transitions which are filled with black (and green) colour is the support of j_1, and the integer "1" on a blacked transition is the number of times that this transition fires in the firing sequence which leads the marking from M_0 to M_0. The net representation of N_{j_1} of j_1 is depicted in Figure 6.6. With the tool Poseidon, it is easy to check that neither a trap nor a co-trap is contained in N_{j_1}. Therefore, the empty marking of the scenario net in the normal-environment context is reproducible according to Theorem 6.6. For example, in the case shown in Figure 6.5,

the empty marking of the scenario net will be reproduced after firing each of the transitions `t1,t2,Receiving MSG,Switch to Other Scenarios` and `t4` once in a proper order (e.g., `t2`→ `t1`→ `Receiving MSG`→ `Switch to Other Scenarios`→ `t4`). Besides, the corresponding entry of the support of j_1 is 1, so the empty marking of the scenario net in 1-configured-environment context is also reproducible according to Proposition 6.7. Furthermore, no co-trap is contained in N_E, thus no dead transition is existing in $(N_E, \emptyset)$ according to Corollary 6.11.

Similarly, other T-invariants of N_E are obtained as in Table 6.3. The amount of input boundary places $|P_I|$ in this case is 3, $\forall t \in T$, $|{}^\bullet t| \leq 2 < |P_I|$, thus no dead transitions were found in the scenario net in 1-configured-environment context ((N_E, M_{01})) according to Proposition 6.14. However, this does not mean that there are no dead transitions in the scenario net in 1-configured-environment context ((N_E, M_{01})) since Proposition 6.14 only provides a sufficient condition for determining dead transitions.

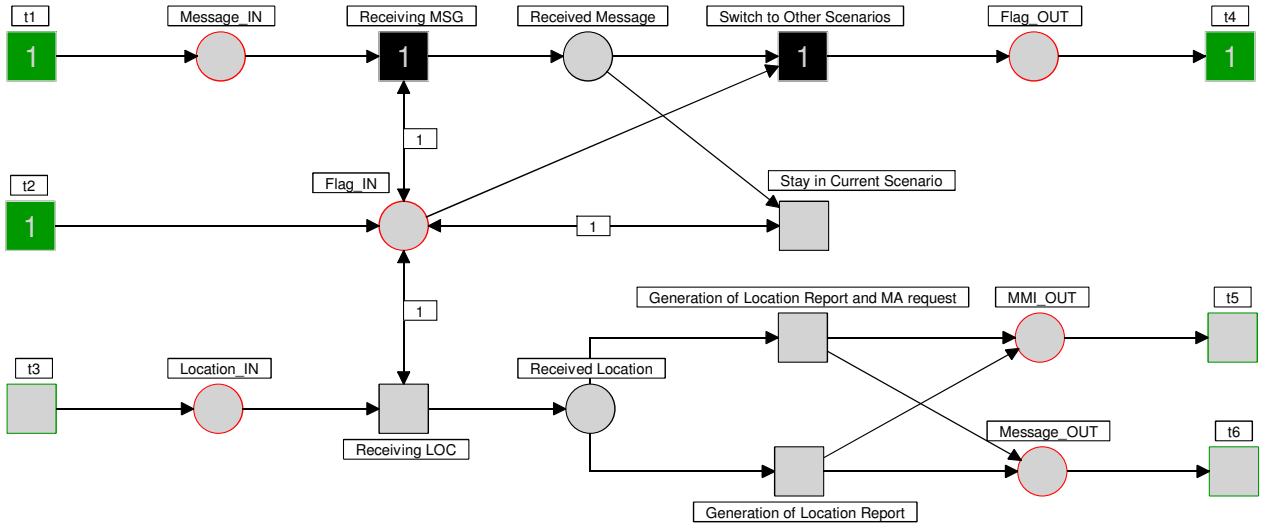


FIGURE 6.5: A T-invariant of the scenario net in normal-environment context

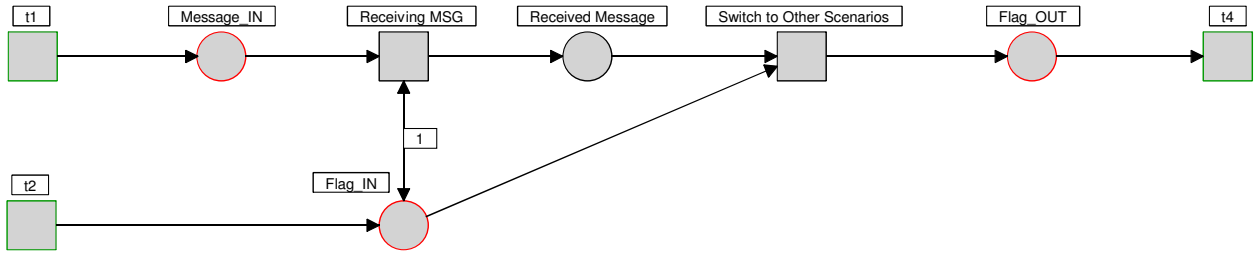


FIGURE 6.6: Net representation of the T-invariant j_1

TABLE 6.3: T-invariants of the net N_E

T-invariants	t1	t2	t3	t4	t5	t6	Receiving MSG	Switch to Other Scenarios	Stay in Current Scenario	Receiving LOC	Generation of Location Report and MA request	Generation of Location Report
j_1	1	1	0	1	0	0	1	1	0	0	0	0
j_2	1	0	0	0	0	0	1	0	1	0	0	0
j_3	0	0	1	0	1	1	0	0	0	1	1	0
j_4	0	0	1	0	1	1	0	0	0	1	0	1

6.5.3 Controllability: Scenario Nets' Ability of Terminating in Empty Markings

In order to demonstrate that the scenario net will terminate in the empty marking, the net in Figure 6.5 can be adapted into an open net in absorbing-environment context shown in Figure 6.7, since the scenario net can only receive messages or location data from the outside environment when the scenario net is activated, i.e., the place `Flag_IN` is marked. It is easy to observe that if no token is on the place `Flag_In`, neither `t1` nor `t3` can fire. Therefore, once the scenario net is terminated, no more tokens on the places `Message_IN` and `Location_IN` will be generated. Besides, it is not difficult to find out that no trap is contained in the open net in absorbing-environment context with the help of the tool, so the scenario net will terminate in the empty state according to Proposition 6.17. In this case, the interfaces of places `Message_IN`, `Location_IN`, `Flag_IN`, `Flag_OUT`, `MMI_OUT` and `Message_OUT` are taken as internal places of the net.

6.5.4 Verification Results

Based on the structural analysis, following conclusions could be drawn:

1. the empty state (marking) of a scenario net of the on-board module is reproducible after processing some incoming data sent by the traffic control centre and the localisation unit;

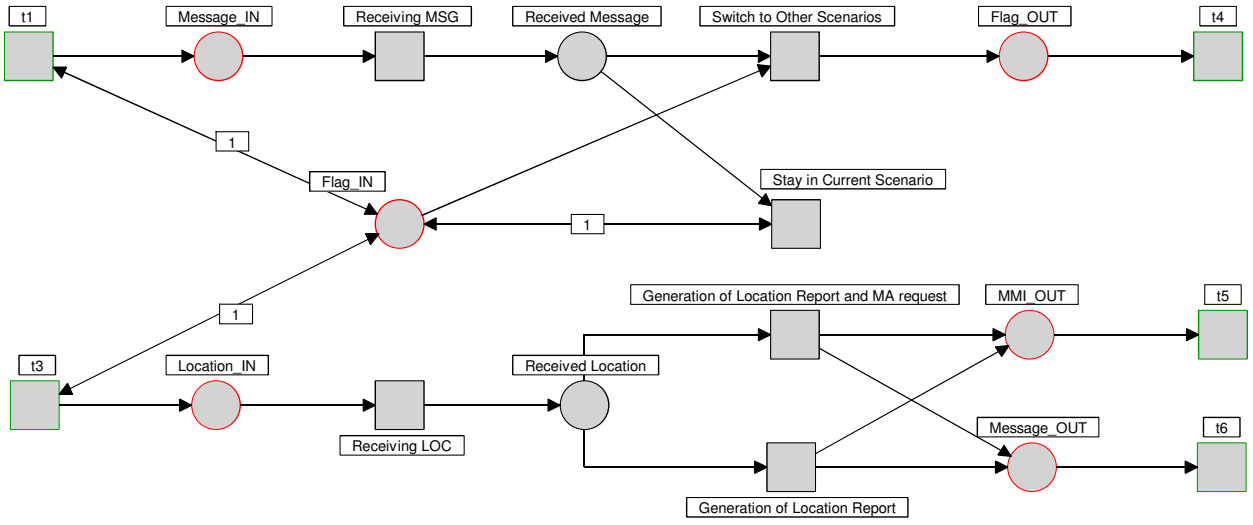


FIGURE 6.7: Scenario net in the absorbing-environment context

2. the empty state of a scenario net is reproducible after processing exactly one set of incoming data. A *set of incoming data* refers to the set of all input boundary places of an open net that each input boundary place is either marked with one token or none;

3. a scenario net will always terminate in the empty state;

4. no dead transition is contained in the scenario nets. In other words, every transition has the possibility to fire.

6.5.5 Discussion

In many cases such as in this thesis, the system model is developed with high-level Petri nets. Structural verifications for high-level models should be careful since they might need to be mapped into the corresponding low-level Petri nets first. An attempt for mapping a CPN module to an open net has been made as follows.

In CPN models, a marking is a function M that maps each place p into a *multiset* [2] of tokens. Considering the multiset of tokens on a place as one abstract token without colour, the mapping ψ from a CPN module Ω_M to an open net N_O is defined as following:

$\Omega_M = (\Omega, T_{sub}, P_{port}, PT)$ where:

1. $\Omega = (P, T, A, \Sigma, V, C, G, E, I)$ is a non-hierarchical Coloured Petri Net.

[2] A multiset m over a non-empty set S can be viewed as a function from S into the set of non-negative numbers $\mathbb{N}$. The function maps each element s into the number of appearances, $m(s)$, of the element s in the multiset m. The non-negative integer $m(s)$ is also called the *coefficient* of s in m[15].

2. $T_{sub} \subseteq T$ is a set of substitution transitions.
3. $P_{port} \subseteq P$ is a set of port places.
4. $PT : P_{port} \rightarrow \{IN, OUT, I/O\}$ is a port type function that assigns a port type to each port place.

$N_O = (P_I, P_O, P', T', F)$ is an open net.

ψ:

1. $(P \setminus P_{port}) \cup P_{port_{I/O}} \rightarrow P''$;
2. $T \rightarrow T''$;
3. $A \rightarrow F''$;
4. $\forall p \in P_{port_{IN}}, p \rightarrow p_i \cup t_{add_i}$ where $p_i \in P_I, {}^\bullet t_{add_i} = p_i, t_{add_i}^\bullet \subset P''$;
5. $\forall p \in P_{port_{OUT}}, p \rightarrow p_o \cup t_{add_o}$ where $p_o \in P_O, {}^\bullet t_{add_o} \subset P'', t_{add_o}^\bullet = p_o$;
6. $T' = T'' \cup P_I^\bullet \cup^\bullet P_O$;
7. $P' = P'' \cup P_I \cup P_O$;
8. $F = F'' \cup (\bigcup(p_i \times t_{add_i})) \cup (\bigcup(t_{add_o} \times p_o))$.

6.6 Summary

This chapter verifies a Petri net model from the perspective of system structure which might affect the behaviour of the system using the notion of open nets. Open nets extend the normal Petri nets with open places (input/output boundary places) which serve as interfaces to their environments. The tokens on the open places represent messages that can be received but not yet received, or messages that are about to be sent out but not yet be sent out. Therefore, open places are given in terms of communication channels. Taking the environments of the open nets in into consideration, three kinds of application contexts are defined, i.e., normal-environment contexts, 1-configured-environment contexts and absorbing-environment contexts. Based on the theory of structural properties of Petri nets, the reproducibility of empty markings of open nets in different environmental contexts is discussed. In order to investigate the consistency of the scenario nets of the on-board subsystem model, a generic scenario net is constructed for applying the theory introduced in this chapter. For verifying the property

of controllability of the scenario nets, the generic scenario net is adapted and the verification is performed by checking that whether the open nets in absorbing-environment contexts will always end up with empty markings.

Chapter 7

Reachability Analysis Based on Net Unfoldings

For the behavioural analysis of a CPN model, state space based methods can be applied initially, e.g., in the CPN Tools [35], standard behavioural properties such as reachability property, boundedness property, home property, liveness property and fairness property can be investigated by asking for a state space report or using query functions. Reachability property, which is the major aspect in system behaviour analysis, investigated by using query functions can only gain the reachability properties of a single marking that is concerned. One cannot have an overview of the reachability property of all the reachable markings when the sequences of transitions of reachable markings are of interest. Instead of using query functions, *reachability graphs* provide all reachable markings if the state space is finite. Nevertheless, the state space of a large-scale system, especially a concurrent system, could be too large to generate the reachability graph. In addition, the state space of a system might be infinite when the system is unbounded [46]. This is known as the *state explosion problem*. To avoid the state explosion problem, McMillan presented a specific technique in [118]. This technique uses the notion of *unfolding*, a partial order [130], [119] semantics of Petri nets, introduced in [131]. He proposed an algorithm to construct a *truncated unfolding*, a finite fragment of the unfolding, which is sufficient to represent all the reachable markings.

In this chapter, the reachability property of the on-board module of the SatZB model established in chapter 3 is discussed based on Petri net unfoldings. First, various techniques for investigating the reachability properties of Petri net models are briefly introduced. Second, based on the formal introduction of Petri net unfoldings, reachability analysis for both 1-safe Petri nets and CP-nets is explored. Last, an application example of the on-board module is provided.

7.1 Introduction

Reachability is a fundamental basis for studying the dynamic properties of any system [46]. Reachability of states is one of the key problems in the area of automatic verification, and most safety properties of systems can be reduced to simple reachability properties [132]. For a safety-critical system such as a train control system, the risk analysis should be carried out at the very beginning of the system life-cycle according to EN50126, which results in a list of hazardous states of the system. Methods of hazard analysis for railway systems have been studied e.g. in [99], [100] and [101]. Given a Petri net model (specification) of a system, the hazardous states can be interpreted into hazardous markings of the model. If the state space (reachability graph) of the system model is generated, then the reachability of hazardous states can be verified. Intuitively, the hazardous markings should be unreachable from any other markings. Apart from verifying the safety aspect of the system, functional verification can also be performed. Each function is mapped into a firing sequence in the Petri net model and each firing sequence corresponds to a path in the reachability graph. Therefore, the functional verification is done by verifying the existence of specific firing sequences in the reachability graph. The processes of safety verification and functional verification by reachability analysis are shown in Figure 7.1.

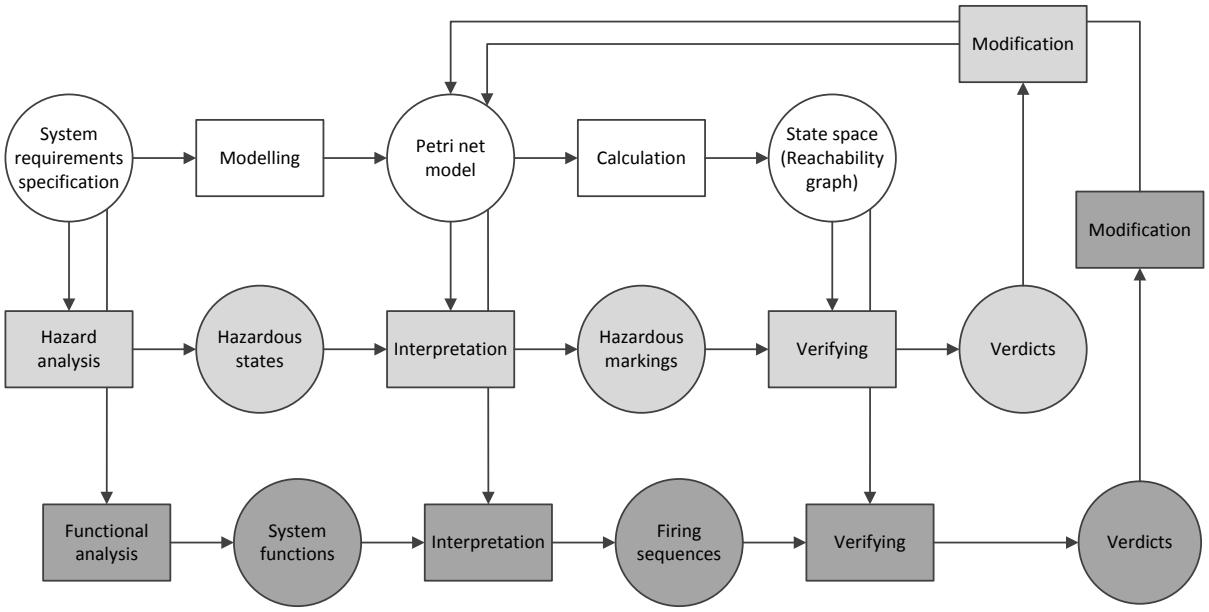


FIGURE 7.1: Reachability analysis for system models

The reachability properties of Petri net models are concerned with determining whether a marking M' is reachable from another marking M, i.e., whether there exists an occurrence sequence starting from M which leads to the marking M'. The main problem of reachability

investigation by using reachability graphs is that it suffers from the state explosion problem. Even a relatively small system model can yield a very large state space, which is mainly caused by the system concurrency. To alleviate this problem, a number of techniques have been proposed. They can be roughly be classified as *avoiding generating a single reachability graph for the system model as a whole, or aiming at an explicit generation of a reduced (though sufficient for a given verification task) representation (e.g., abstraction and partial order reduction techniques)* [133]. Some of these techniques are briefly introduced in the following subsections.

7.1.1 Hierarchical Reachability Graph Techniques

To cope with the state explosion problem, the authors of [134] proposed a *divide-conquer* method. In this method, a Petri net is regarded as a collection of subnets communicating or synchronising with each other only through transitions. First, the reachability graphs (called *R*-graphs here) of subnets, which are the lowest level in the hierarchical representation of the state space, are generated. These reachability graphs are compressed by aggregating *I-O similar* equivalence states, and combined by synchronising transitions to obtain higher level reachability graphs. This compress-combine operation is repeated until the state space is combined and condensed into one *R*-graph, which is the highest level in the hierarchy.

The reachability property of a Petri net consisting of n subnets, could be investigated by analysing the hierarchical reachability graph (HRG, a tree with n leaf nodes). If a marking M is reachable from the initial marking M_0, then M is mapped to a state (corresponding vertex) of the root node.

However, the HRG method has restrictions. It is only applicable to *safe or any bounded nets consisting of subnets synchronised only through transitions*, which means that only the reachability graph of a Petri net is able to be constructed, can this method be applied. It can not ensure the improvement of the reachability analysis comparing to conventional methods. The amount of reduction that can be obtained in the HRG or the efficiency of the HRG depends on the structure of the net.

7.1.2 Modular State Space Techniques

For the purpose of state space analysis of industrial size systems that consist of a set of modules, avoiding the construction of a single state space of the entire system, a *Modular State Space* method is proposed in [135]. The behaviour of the total system is captured by the state spaces of modules combined with a *Synchronisation Graph*. A Synchronisation Graph is constructed by the state spaces of modules that *share only transitions composing the entire system*.

For the system constructed with modules sharing places, a transformation from a modular CP-net using place fusion to a modular CP-net using only transition fusion is defined by the authors.

The modular construction of the occurrence graph allows either to gain time or space, and the efficiency gained depends a lot on the modular structure chosen. The properties of the system can be checked directly on the Modular State Space.

7.1.3 Reachability Investigation by Standard Queries

For CP-nets, *state spaces*, also called occurrence graphs, reachability graphs or reachability trees [95], calculate all reachable states (markings) and state changes (occurring binding elements) of the CPN model and represent these in a directed graph where the nodes correspond to the set of reachable markings and the arcs correspond to occurring binding elements. Therefore a state space is a directed graph where we have a node for each reachable marking and an arc for each occurring binding element. State spaces can become very large, so they need to be calculated and analysed by means of a computer-aided tool such as CPN Tools with which the State Space Tool (SS tool) [95] is integrated. However, in some cases, a state space is still too big to calculate by a tool due to the state explosion problem or even if the state space is calculable, it will take too much time.

In [15], simple reachability properties are investigated by using the standard query function `Reachable`. The function `Reachable` takes a pair (n, n') of integers as an argument and checks whether there exist a path in the state space leading from node n to node n'. For example, the query `Reachable (1,5)` checks whether the marking represented by node 5 is reachable from the marking represented by node 1. If it does, then the function returns true. The function also has the chatty version `Reachable'` [95], which returns the same result together with an explanation of an occurrence sequence (directed path) of minimal length. Additionally, the function `SccReachable` returns the same result as `Reachable`, but it uses the SCC graph (Strongly-Connected-Component graph). The function `AllReachable` determines whether all the reachable markings are reachable form each other.

Beside checking whether there exist a path in the state space leading from node n represented by an integer to node n' represented by another integer, it is also possible to check whether there exist a path in the state space leading from a specified marking to another specified marking. First we implement a predicate `DesiredTerminal`, which given a node `n`, returns `true` if the marking represented by `n` corresponds to the desired terminal state. And then we can obtain a list of nodes containing those markings that satisfy the predicate `DesiredTerminal` by using the standard query function `PredAllNodes`, which takes a

predicate as an argument. The query is as following: `PredAllNodes DesiredTerminal`. Last we can choose the nodes from the lists returned by the functions `PredAllNodes` as the arguments of the functions `Reachable` or `Reachable'`.

7.1.4 Unfolding-based Techniques

The authors of [132] studied four unfolding-based techniques to the reachability problem for 1-safe Petri nets. The authors define the reachability problem as follows: *given a set of places of the net, determine if some reachable marking put a token in all of them*.

The first method introduced by Melzer [136] is a method for checking the reachability of a marking based on linear programming. The basic concept of this method is the so-called *marking equation* that can be used as an algebraic representation of the set of reachable markings of an acyclic net (e.g. unfolding). The second method introduced by Heljanko [137] is a method for reachability checking of complete finite prefixes using logic programs with stable model semantics. The main idea of this approach is to translate the problem into a rule based logic program and to check if there exists a stable model. The third method introduced by the authors is a new graph theoretic algorithm, which uses the *co-relation* defined on the set of conditions of a prefix. Based on a theorem in [136] to checking the reachability of a *partial marking*, the solution of reachability problem is transformed to finding a *k-clique* in a *k-partite graph*. To make the method also work for the partial marking that includes places which should not carry a token, *complementary place* is introduced. The last method is the *on-the-fly* verification introduction by McMillan [117]. The main idea of this method is to insert a new transition t_{new} into the original net such that its preset is the partial marking to be checked. If there exists an event in the prefix that labelled with t_{new}, then the partial marking is reachable.

7.2 Reachability Analysis Based on Net Unfoldings

According to the introduction of the techniques proposed to alleviate the sate space explosion problem, either the hierarchical reachability graph techniques or the modular state space techniques requires that the subnets only share transitions composing the entire system. If a Petri net model constructed with modules sharing places, a transformation from a modular Petri net using place fusion to a modular Petri net model using only transition fusion is needed. The CPN model of SatZB established in this thesis is composed by sharing places, so a transformation to a transition shared system is inevitable if we apply these two techniques. This is however an effort-consuming process, and more importantly, an error-prone process,

in particular, for a complex system since the transformation is done by hand and there is no implementation of algorithms known by now according to the author's knowledge. Reachability investigation by standard queries only works for simple reachability properties. It is not able to provide all the occurrence sequences leading the marking from one to the other. As a result, we choose unfolding-based techniques to analyse the reachability property of the SatZB model. In [132], linear programming, logic programs with stable model semantics or co-relation is explored to check the reachability of a marking, while in this chapter, the concept of *configuration* is used.

7.2.1 Unfoldings of Low-level Petri Nets

On the basis of the basic definitions and notations of Petri nets introduced in chapter 2, we present the notation and basic definitions on unfoldings as follows relying on [15], [33], [118], [119], [74], [39], [40], [124], [130].

7.2.1.1 Labelled Nets

Let Γ be an alphabet. A *labelled net* is a pair (N, ℓ), where $N = (P, T, F)$ is a Petri net and ℓ: $P \cup T \to \Gamma$ is a labelling function. Note that different nodes (places and transitions) of the net can carry the same label.

7.2.1.2 Occurrence Nets

Given two nodes x and y of a Petri net. x is *causally related* to y, denoted by $x < y$, if there is a (possibly empty) path of arrows from x to y; x and y are in *conflict*, denoted by $x\#y$, if there is a place z, different from x and y, from which one can reach x and y, exiting z by different arrows; x and y are *concurrent*, denoted by $x \; co \; y$, if neither $x < y$ nor $y < x$ nor $x\#y$ hold. A *co-set* is a set of nodes X such that $x \; co \; y$ for very $x, y \in X$.

An *occurrence net* is an Petri net $ON = (B, E, F)$ where B is the set of *conditions* (places), E is the set of *events* (transitions) and F is a flow relation, satisfying the following three properties:

- $\forall b \in B, |{}^\bullet b| \leq 1$, i.e., a condition has only one or none input event;
- F is acyclic, i.e., the net seen as a directed graph, has no cycles;
- $\forall x \in B \cup E, \neg(x\#x)$, i.e., no node is in self-conflict.

A place b ($b \in B$) of an occurrence net is *minimal* if ${}^\bullet b = \emptyset$, i.e. it has no input transitions. The *default initial marking* of an occurrence net puts one token on each minimal place. $Min(ON)$ denotes the set of minimal elements of $B \cup E$.

7.2.1.3 Branching Processes

A *homomorphism* from an occurrence net $ON = (B, E, F)$ to a Petri net system $\Sigma = (P, T, F, M_0)$ is a mapping $h : B \cup E \to P \cup T$ such that

- $h(B) \subseteq P$ and $h(E) \subseteq T$, i.e., conditions are mapped to places, and events to transitions;
- $\forall e \in E$, the restriction of h to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and ${}^\bullet h(t)$, and similarly for $t^\bullet$ and $h(t)^\bullet$, i.e., transition environments are preserved;
- The restriction of h to $Min(ON)$ is a bijection between $Min(ON)$ and M_0, i.e., minimal conditions are mapped to the initial marking;
- $\forall e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $h(e_1) = h(e_2)$, then $e_1 = e_2$, i.e., there is no redundancy.

A *branching process* of a Petri net system $\Sigma = (P, T, F, M_0)$ is a pair $\beta = (ON, h)$ such that ON is an occurrence net and h is a homomorphism from ON to Σ. As a matter of fact, a branching process is a labelled net.

7.2.1.4 Unfoldings and Prefixes

Two branching processes $\beta_1 = (ON_1, h_1)$ and $\beta_2 = (ON_2, h_2)$ of a Petri net system Σ are *isomorphic* if there is a bijective homomorphism h from ON_1 to ON_2 such that $h_2 \circ h = h_1$. Intuitively, two isomorphic branching processes differ only the names of conditions and events.

There exists a unique maximal branching process up to isomorphism for a Petri net system Σ, called the *unfolding* of Σ. Loosely speaking, the unfolding of a Petri net system is a labelled Petri net, more precisely a Petri net whose places and transitions are labelled with places and transitions of the original net.

Let $\beta = (ON, h)$ and $\beta' = (ON', h')$ be two branching process of a Petri net system Σ. β is a *prefix* of β' if ON is a subnet of ON' satisfying:

- if a condition belongs to ON, then its input event in ON' also belongs to ON;

- if an event belongs to ON, then its input and output conditions in ON' also belong to ON.

and h is the restriction of h' to ON.

The unfolding of a Petri net system Σ is the union of all branching process of Σ. Every branching process is a *prefix* of the unfolding.

7.2.1.5 Configurations and Cuts

A *configuration* of an occurrence net is a set of events C satisfying the following to properties:

- $\forall e \in C : e' < e \Rightarrow e' \in C$, i.e., C is causally closed;
- $\forall e, e' \in C : \neg(e \# e')$, i.e., C is conflict-free.

Given an event $e \in E$, the configuration $[e] = \{e' \in E | e' \leq e\}$ is called the *local configuration* of e, and $\langle e \rangle = [e] \setminus \{e\}$ denotes the set of *causal predecessors* of e.

A maximal co-set $B' \subseteq B$ with respect to set inclusion is called a *cut*. Let C be a finite configuration of a branching process $\beta = (ON, h)$, then the co-set $Cut(C)$, defined as following, is a cut: $Cut(C) = (Min(ON) \cup C^{\bullet}) \setminus^{\bullet} C$. In particular, the set of place $h(Cut(C))$ is a reachable marking, denoted by $Mark(C)$, namely $Mark(C) = h(Cut(C))$.

7.2.1.6 Possible Extensions and Cut-off Events

A branching process β of a Petri net system Σ is *complete* if for every reachable marking M there exists a configuration C in β such that:

- $Mark(C) = M$, i.e., M is represented in β;
- $\forall t \in T$ in Σ enabled by M, there exists a configuration $C \cup \{e\}$ in β such that $e \notin C$ and e is labelled by t.

The unfolding of a Petri net system is always complete. A *complete prefix* of the unfolding contains as much information as the unfolding, in the sense that one can construct the unfolding from it. However, the unfolding of a Petri net system, as well as its complete prefix could be infinite whenever the Petri net system has infinite runs, i.e., there exist cycles in the reachability graph of the Petri net system. To deal with this problem, *finite complete prefixes*

of the unfolding of a Petri net system that are sufficient to represent certain information, e.g., all reachable markings, or decide a certain behavioural property, e.g., deadlock freeness, are defined by truncating the unfolding. For this purpose, *possible extensions* and *cut-off events* should be declared first.

Given a configuration C, the fact that $C \cup E$ is a configuration such that $C \cap E = \emptyset$, denoted by $C \oplus E$, is an *extension* of C, and E is a *suffix* to C. A *possible extension* of a branching process β of a Petri net system Σ is a pair (t, X), where $t \in T$ in Σ, and $X \subseteq B$ is a co-set in ON of β such that:

- $h(X) = {}^\bullet t$;
- (t, X) is not a part of β.

A *partial order* $\prec$ on the finite configurations of the unfolding of a Petri net system is an *adequate order* if:

- $\prec$ is well-founded, i.e., it has no infinite descending chains;
- $C_1 \subset C_2$ implies $C_1 \prec C_2$;
- $\prec$ is preserved by finite extensions, i.e., if $C_1 \prec C_2$ and $Mark(C_1) = Mark(C_2)$, then the isomorphism f [1] satisfies $C_1 \oplus E \prec C_2 \oplus f(E)$ for all finite extension $C_1 \oplus E$ of C_1.

Let β be a branching process and $\prec$ be an *adequate partial order* on the configuration of β. An event e is a *cut-off event* (with respect to $\prec$) if β contains a local configuration $[e']$ such that:

- $Mark([e]) = Mark([e'])$;
- $[e'] \prec [e]$.

7.2.1.7 McMillan's Unfolding Algorithm

In [118] McMillan introduced an unfolding algorithm as follows:

1. For each of the initial tokens, make a copy of the place on which it resides in the occurrence net (label the copies appropriately).

[1] f is a mapping from the finite extensions of C_1 onto the finite extensions of C_2; the image of $C_1 \oplus E$ under this mapping is $C_2 \oplus f(E)$.

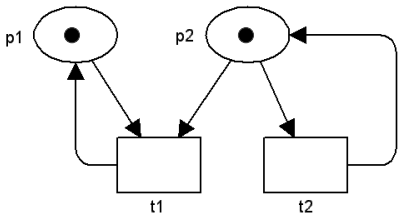


FIGURE 7.2: A Petri net system Σ_1

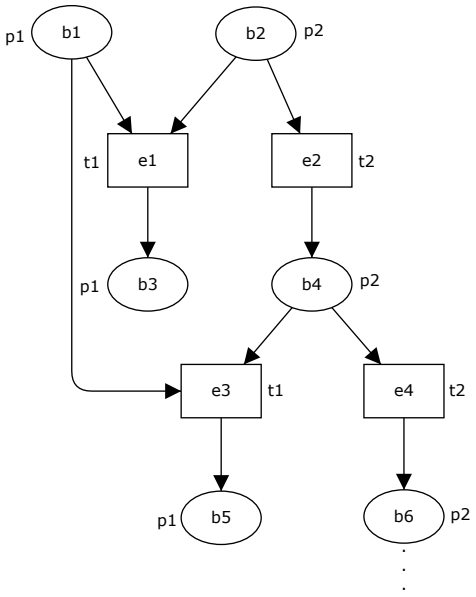


FIGURE 7.3: The unfolding of Σ_1

2. Choose a transition from the Petri net and call it t.

3. For each place in the preset of t, find a copy in the occurrence net and mark it with a token (if one cannot find a copy, go back to step 2). For a given t, do not choose the same subset of places in the occurrence net twice.

4. If any of the places you marked are not concurrent, go to step 2.

5. Make a copy of t in the occurrence net and call it t'. Draw an arrow from every place you marked in the occurrence net to t'. Erase the tokens.

6. For each place in the postset of t, make a copy in the occurrence net (label the copies appropriately) and draw an arrow from t' to it.

An example of applying McMillan's unfolding algorithm is shown in Figure 7.2 and Figure 7.3. Figure 7.2 is a Petri net system Σ and Figure 7.3 is the initial part of the unfolding of Σ, which is infinite. Because conditions b_3 and b_4 are not concurrent, i.e., conflict, the input of the event e_3 should be conditions b_1 and b_4 instead of conditions b_3 and b_4 according to step 4.

However, the process of unfolding a Petri net with the algorithm presented above could be infinite. McMillan thus adds one more step between steps 4 and 5 to generate *truncated unfolding* of the Petri net later on:

4.5. Make a copy of t in the occurrence net and call it t'. If t' is a cut-off event, go to step 2.

It is easy to observe that events e_3 and e_4 are cut-off events in Figure 7.3.

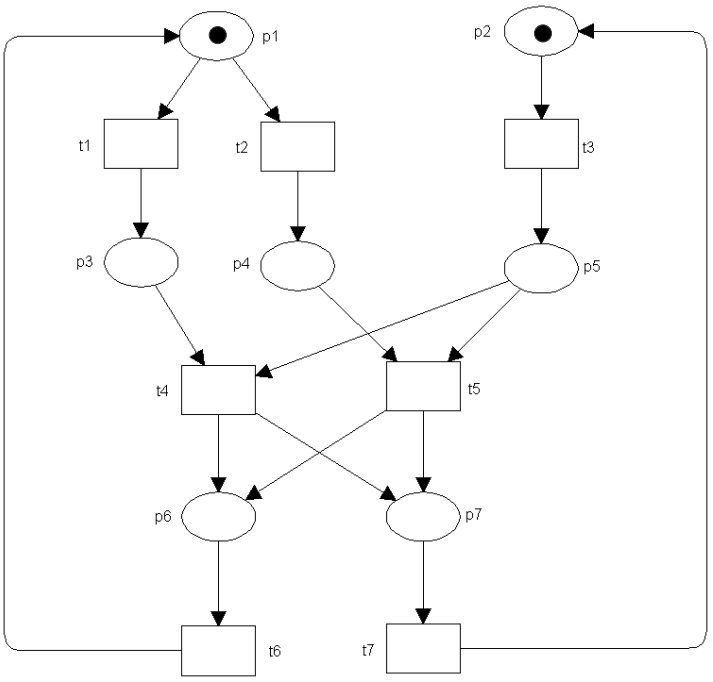


FIGURE 7.4: A Petri net system Σ_2

A more complicate example of unfolding a Petri net system is shown in Figure 7.4 and Figure 7.5. In Figure 7.5, set $\{e_1, e_3, e_4, e_6\}$ is a configuration, while set $\{e_1, e_4\}$ (not causally closed) and $\{e_1, e_2\}$ (not conflict-free) are not. Let $C = \{e_1, e_3, e_4, e_6\}$, then $Cut(C) = \{b_{10}, b_7\}$ and $Mark(C) = h(Cut(C)) = \{p_1, p_7\}$. A configuration of an unfolding could have multiple *linearisations* , and each linearisation is a firing sequence of the occurrence net (from the default initial marking) containing each event from the configuration exactly once, and no further events. All linearisations of a configuration lead to the same reachable marking. The configuration $\{e_1, e_3, e_4, e_6\}$ has two linearisations $e_1\ e_3\ e_4\ e_6$ and $e_3\ e_1\ e_4\ e_6$. All these two linearisations lead to the reachable marking $\{p_1, p_7\}$. Since $Mark([e_{16}]) = Mark([e_4]) = \{p_6, p_7\}$ and $[e_4] \prec [e_{16}]$, e_{16} is a cut-off event, and so do e_{17}, e_{18}, e_{19}.

McMillan's algorithm of constructing a truncated unfolding, a finite fragment of the unfolding which is sufficient to represent all the reachable markings. In [119], based on the fact that the truncated unfolding, called *finite complete prefix* here, generated from McMillan's algorithm may be much larger than necessary (exponentially larger in the worst case), the authors provide a refinement of McMillan's algorithm which generates a minimal complete prefix. This prefix is always smaller than or as large as the prefix generated with McMillan's algorithm. The refinement of McMillan's algorithm is based on the semantics of *adequate total order* while McMillan's algorithm belongs to *adequate partial order* [119]. Nevertheless, we use McMillan's algorithm in this thesis for convenience sake, so the details of the refinement of McMillan's algorithm will not be discussed here.

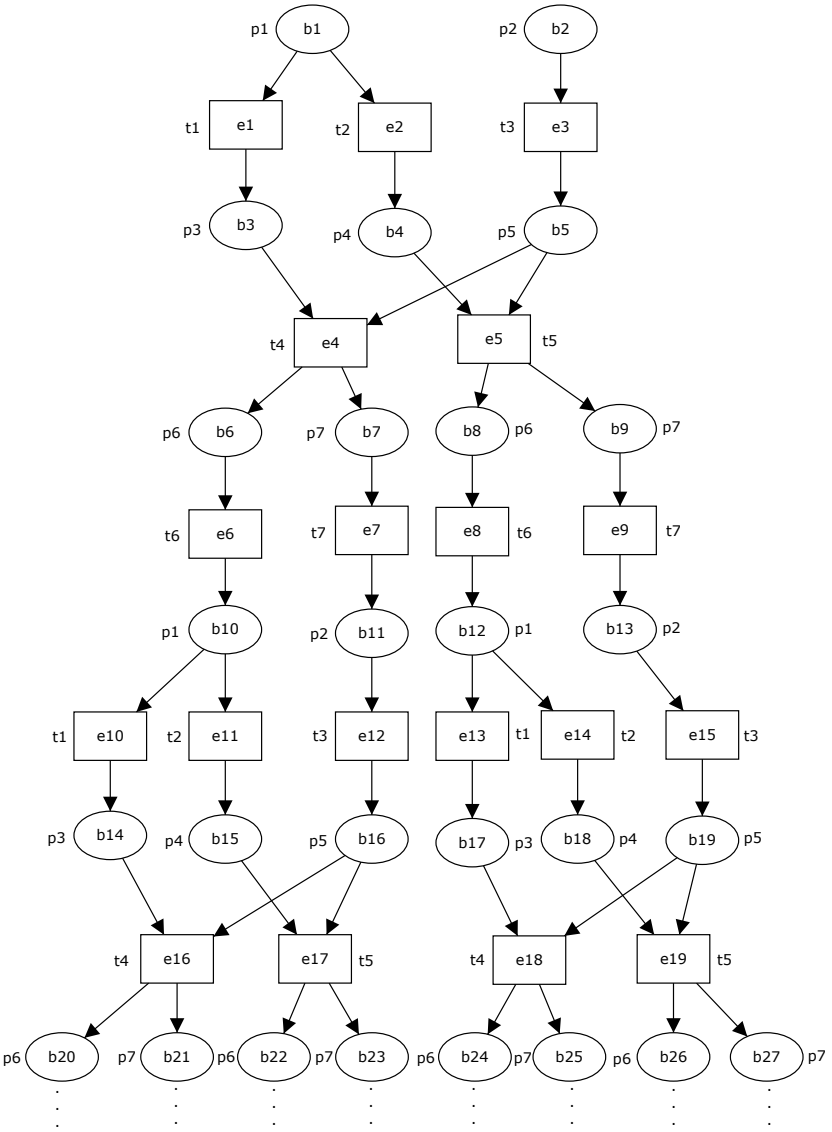


FIGURE 7.5: The unfolding of Σ_2

7.2.2 Unfoldings of Coloured Petri Nets

This thesis suggests to modularise large-scale systems with the hierarchical structuring mechanism of CP-nets, which are high-level Petri nets. Therefore, unfolding high-level Petri nets with hierarchical structure is a realistic issue for investigating the behavioural properties of high-level Petri net models. In [74], the authors proposed an approach to build a finite complete prefix directly from a high-level Petri net, avoiding potentially expensive translation into a low-level one. As the high-level Petri net model, the *M-nets* [72] is adopted by the authors. The approach is conservative in the sense that all the verification tools employing the traditional unfolding can be reused with prefixes derived directly from high-level Petri nets. Besides, In [39], the author has defined branching processes of CP-nets based on the definition of branching processes and unfolding of high-level Petri nets presented in [74] and

[138]. In addition, he discusses the two ways of unfolding CP-nets: unfolding per transformation and direct unfolding using McMillan's unfolding algorithm [118]. However, one of the restrictions he raised in his approach is that it is restricted to *non-hierarchical* CP-nets. To unfold a high-level Petri net with hierarchical structure, a extension of [39] is achieved in the following by unfolding a hierarchical CP-net.

7.2.2.1 Unfoldings of Non-hierarchical CP-nets

Based on the unfolding theory of low-level Petri nets as discussed above, it is easy to get the idea of unfolding CP-nets as the following: first transform the CP-nets into low-level Petri nets; and then unfold the transformed low-level Petri nets, which could be done by employing e.g. McMillan's algorithm (see [118]) or the improved algorithm in [119]. The rules for transforming CP-nets into low-level Petri nets are presented in [39]. However, there are some drawbacks to unfold a CP-net by transforming into a low-level Petri net according to the mentioned rules. On the one hand, it is impossible to transform a CP-net with infinite sets of colours. On the other hand, the transformation may generate a large number of low-level Petri net places and transitions, that will never get marked and never get enabled. This leads to a huge intermediate low-level Petri net. Considering these issues, Januzaj [39] proposed an approach of unfolding a non-hierarchical CP-net $\Omega = (P, T, A, \Sigma, V, C, G, E, I)$ directly without transforming into intermediate low-level Petri nets using McMillan's unfolding algorithm as following:

1. Transform each token element (p, c) [2] of the current marking into a condition b and label it with (p, c);
2. Choose a transition t;
3. For each place $p \in^\bullet t$, find all possible copies of p in the occurrence net, i.e., all conditions b labelled by (p, c), and mark them with c. If no such copy can be found, go to step 2. Note that for a chosen t, do not choose the same subset of places in the occurrence net twice.
4. For each possible binding b_p of $B(t)$ [3], regarding the current marking, create a binding element (t, t_p) and apply the following:
   4.1. transform (t, t_p) into an event e and label it with t;

[2] A token element is a pair (p, c), where p is a place and $c \in C(p)$ is a colour which may reside on p [15].
[3] $B(t)$ represents the set of all bindings of t.

4.2. for each $M(p), p \in t^\bullet$, generated after the occurrence of the actual binding element (t, b_p), apply step 1;

4.3. connect e to its preset and postset accordingly [4].

5. Go to step 2.

Additionally, the definition of branching processes of CP-nets is also presented in [39] as follows.

A *homomorphism* from a occurrence net $ON = (B, E, F)$ to a CP-net Ω is a mapping $h : B \cup E \rightarrow TE \cup BE$ such that:

- $h(B) \subseteq TE$ and $h(E) \subseteq BE$, i.e., conditions are mapped to token elements and events are mapped to binding elements;
- $\forall e \in E, h(^\bullet e)_{MS} = {}^\bullet h(e)$ and $h(e^\bullet)_{MS} = h(e)^\bullet$, i.e., the environments of binding elements are preserved;
- $h(Min(ON))_{MS} = M_0$, i.e., minimal conditions are mapped to the initial marking;
- $\forall e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $h(e_1) = h(e_2)$, then $e_1 = e_2$, i.e., there is no redundancy.

A *branching process of a CP-net* Ω is a pair $\beta = (ON, h)$ such that ON is an occurrence net and h is a homomorphism from ON to Ω.

Besides, the possible extensions of branching process of CP-nets is defined in [39]:

A *possible extension of a branching process* $\beta = (ON, h)$ *of a CP-net* Ω is a pair $((t, b), X)$, where X is a co-set in β and $(t, b) \in BE(t)$ is an enabled binding element such that:

- $h(X_{MS}) = {}^\bullet t$;
- $((t, b), X)$ is not a part of β.

7.2.2.2 Unfoldings of Hierarchical CP-nets

Januzaj's approach is restricted to non-hierarchical CP-nets. To extend his approach to hierarchical CP-nets, some basic concepts required are presented in the following based on [15]:

[4] It is very important to note that each binding element (t, b) must satisfy the guard $G(t)$, and the preset of each event e must be a co-set, otherwise e should not be generated.

The concept of *modules* in CP-nets is based on a hierarchical structuring mechanism. A module exchanges tokens with its environment (i.e., the other modules) through interfaces, which are places that represent *input ports, output ports or input/output ports*. the module will import tokens via the input port and export tokens via the output port. An input/output port is a port through which a module can both import and export tokens.

In CP-nets, a module is usually represented by a *substitution transition* in its superior hierarchical level. A substitution transition has a rectangular substitution tag positioned next to it. The substitution tag contains the name of a *submodule* which is related to the substitution transition.The input places of substitution transitions are called *input sockets*, and the output places are called *output sockets*. The socket places of a substitution transition constitute the interface of the substitution transition. To obtain a complete hierarchical model, it must be specified how the interface of each submodule is related to the interface of its substitution transition. This is done by means of a *port-relation*, which relates the port places of the submodule to the socket places of the substitution transition. Input ports are related to input sockets, output ports to output sockets, and input/output ports to input/output sockets. When a port and a socket are related, the two places constitute two different views of a single place. This means that related port and socket places always share the same marking and hence conceptually become a single *compound place*. If a port place does not have an initial marking expression, then it obtains its initial marking from the related socket place.

It has been shown in previous paragraph how modules can exchange tokens via port and socket places. It is also possible for modules to exchange tokens via *fusion sets*. Fusion sets allow places in different modules to be glued together into one compound place across the hierarchical structure of the model. The places that are members of a fusion set are called *fusion places* and represent a single compound place, in a way similar to that for a related port and socket place. This means that all instances of all places in a fusion set always share the same marking and that they must have identical colour sets and initial markings.

Unfolding via Transformation. Since it is valid to substitute a substitution transition with its related submodule, a hierarchical CP-net can always be transformed [5] into an equivalent non-hierarchical CP-net with the same behaviour using a process presented in [15]. This process consists of three following steps:

1. Replace each substitution transition with the content of its associated submodule such that related port and socket places are merged into a single place;

[5] In [15] the word "unfold" is used. Anyway, "unfold" has another specific meaning in this paper, so here we use "transform" instead.

2. Collect the content of all resulting *prime modules* into a single module. Note that prime modules are the roots of the module hierarchy;

3. Merge the places in each fusion set into a single place.

After transforming the hierarchical CP-nets into non-hierarchical ones, Januzaj's approach of unfolding CP-nets, either unfold via intermediate low-level Petri nets or direct unfolding, can be adopted. Nevertheless, this method would lead to a large non-hierarchical CP-net, which is the exact model we are trying to avoid by constructing hierarchical CP-net. And an even larger intermediate low-level Petri net model could be yielded if one transforms the transformed non-hierarchical CP-net into low-level Petri nets. Thus obtaining the unfolding of a hierarchical CP-net directly from hierarchical CP-nets is desirable.

Direct Unfolding. On the basis of the approach of unfolding non-hierarchical CP-nets directly from their high-level state presented in [39], an extended approach that unfold a hierarchical CP-net $\Omega_H = (S, SM, PS, FS)$ [6] directly from its hierarchical state is proposed as following:

1. Transform each token element (p, c) of the current marking of the current module $s \in S$ (normally begin with the top abstraction level of the CP-net) into a condition b and label it with (p, c);

2. Choose an ordinary transition t in current module. If there is no such transition t in the current module that can be chosen regarding that for a chosen t, do not choose the same subset of places in the occurrence net twice, then choose a substitution transition t_{sub}, and go to step 6, otherwise end;

3. For each place $p \in {}^\bullet t$, find all possible copies of p in the occurrence net, i.e., all conditions b labelled by (p, c). If no such copy can be found, go to step 2. Notice that for a chosen t, do not choose the same subset of places in the occurrence net twice.

4. For each possible binding b_p of $B(t)$, regarding the current marking, create a binding element (t, t_p) and apply the following:

   4.1. transform (t, t_p) into an event e and label it with t;

   4.2. for each $M(p), p \in t^\bullet$, generated after the occurrence of the actual binding element (t, b_p), apply step 1;

[6] This formal definition is defined in [15] where S is a finite set of modules; $SM : T_{sub} \to S$ is a submodule function that assigns a submodule to each substitution transition; PS is a port-socket relation function that assigns a port-socket relation $PS(t) \subseteq P_{sock}(t) \times P_{port}^{SM(t)}$ to each substitution transition t; $FS \subseteq 2^P$ is set of non-empty fusion sets. More details can be found in [15].

4.3. connect e to its preset and postset accordingly ;

5. Go to step 2;

6. Unfold a submodule $s_{sub} \in S$ assigned to the substitution transition t_{sub}:

   6.1. For each input socket p_{inSock} of the submodule s_{sub}, find all possible conditions b labelled by (p_{inSock}, c) in the occurrence net; for each fusion place p_{fs}, find the condition b labelled by $(p_{fs'}, c)$ in the occurrence net, where $p_{fs'}$ is a member of the fusion set $fs \in FS$ that p_{fs} belongs to (similar to step 1);

   6.2. Choose an ordinary transition t in submodule s_{sub}. If there is no such transition t in the current module can be chosen regarding that for a chosen t, do not choose the same subset of places in the occurrence net twice, go to step 7;

   6.3. For each place $p \in {}^\bullet t$, find all possible copies of p in the occurrence net, i.e., all conditions b labelled by (p, c). If no such copy can be found, go to step 6.2. Notice that for a chosen t, do not choose the same subset of places in the occurrence net twice;

   6.4. For each possible binding b_p of $B(t)$, regarding the current marking, create a binding element (t, t_p) and apply the following:

      6.4.1. transform (t, t_p) into an event e and label it with t;

      6.4.2. for each $M(p), p \in t^\bullet$, generated after the occurrence of the actual binding element (t, b_p), apply step 1;

      6.4.3. connect e to its preset and postset accordingly;

   6.5. Go to step 6.2;

7. For each output socket $p_{outSock}$ of the submodule s_{sub}, find all possible conditions b labelled by $(p_{outSock}, c)$ in the occurrence net; for each fusion place p_{fs}, find the condition b labelled by $(p_{fs'}, c)$ in the occurrence net, where $p_{fs'}$ is a member of the fusion set $fs \in FS$ that p_{fs} belongs to;

8. Go to step 2.

Remark 7.1. The unfolding approach described above assumes that the hierarchical CP-net has only two abstraction levels. Anyway, a hierarchical CP-net with more than two abstraction levels could be unfolded easily by adapting this approach. In addition, the process from step 1 to step 5 is an approach of unfolding non-hierarchical CP-nets.

To exemplify the described approach, a hierarchical CP-net, whose top page is depicted in Figure 7.6 and its submodule assigned to the substitution transition T_{sub} is shown in Figure 7.7, is unfolded. Declaration for the type of the places could limit the reachable markings to finite. This is also could be done by adding guards to corresponding transitions.

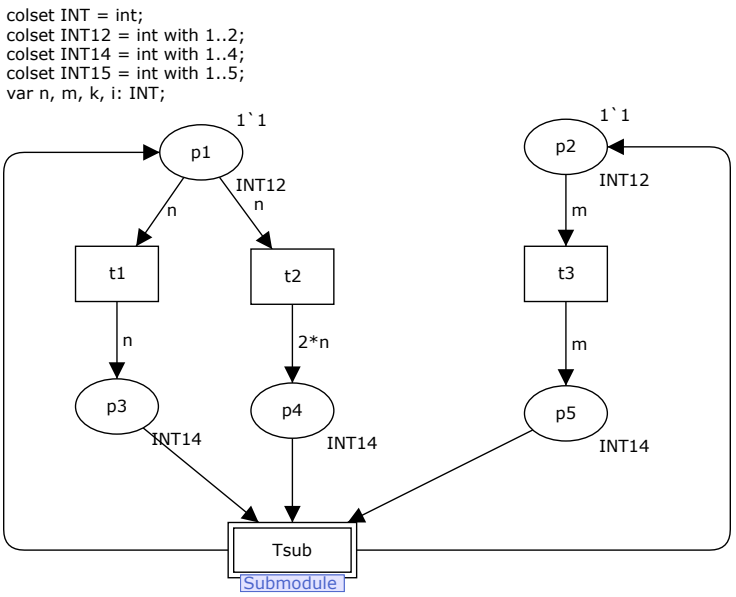


FIGURE 7.6: Top page of a hierarchical CP-net

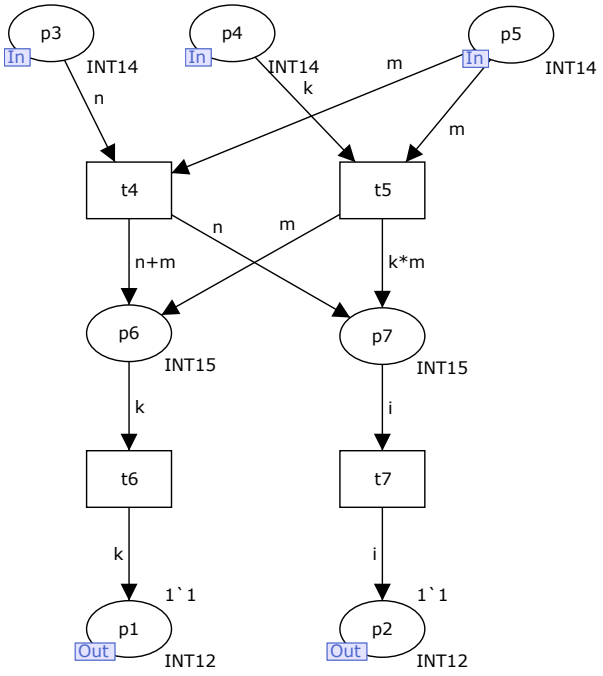


FIGURE 7.7: Submodule ralated to substitution transition T_{sub}

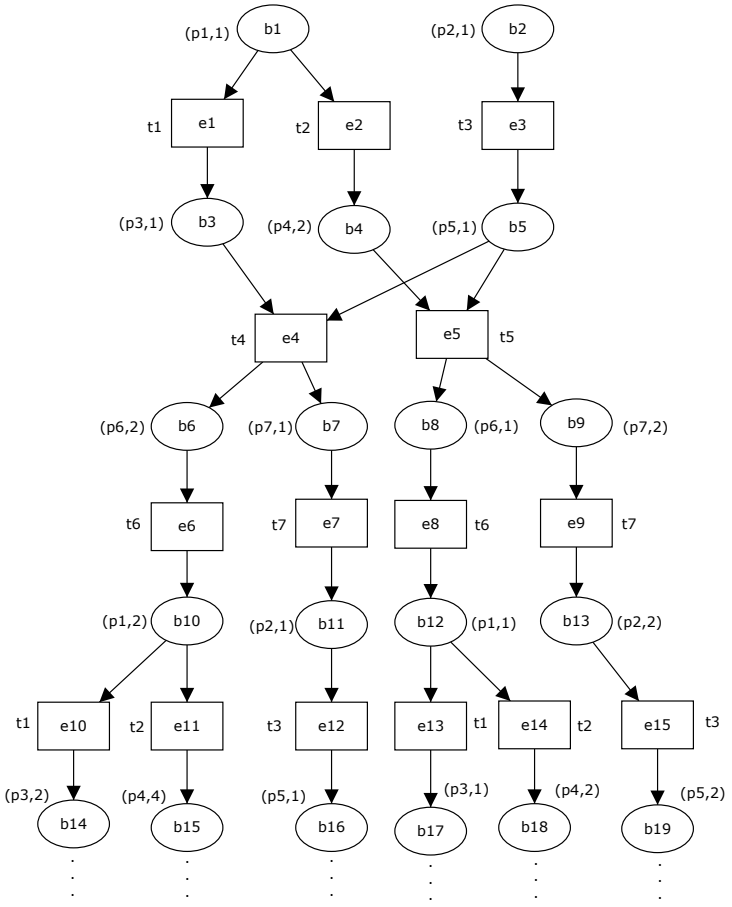


FIGURE 7.8: Unfolding of the hierarchical CP-net

Figure 7.8 illustrates the unfolding of the hierarchical CP-net showing in Figure 7.6 and Figure 7.7. Firstly, the initial part of the unfolding from condition b_1 to b_5 could be generated according to step 1 to step 5; secondly, the part of the unfolding from condition $b3$ to $b13$, which is related to the substitution transition T_{sub}, is constructed based on step 6 and step 7; thirdly, the remaining part could be finished according to step 8 which leads to step 2. So it is a cyclic process of unfolding a hierarchical CP-net no matter how many abstraction level the CP-net has (In this case, there are only two abstraction levels.). Figure 7.8 just shows a initial part (finite prefix) of the unfolding due to saving space, even though the unfolding of the given CP-net is finite.

In fact, this is a *up-down-up* approach with respect to the order of unfolding the abstraction levels of the hierarchical CP-nets. Our approach adopts McMillan's unfolding algorithm as it is based on Januzaj's approach, which uses McMillan's unfolding algorithm.

7.2.3 Reachability Analysis for 1-safe Petri Nets

7.2.3.1 Reachability Problems

A marking M of a 1-safe Petri net is a mapping $P \to \{0,1\}$, i.e., $\forall p \in P, M(p) = 0$ or $M(p) = 1$. To describe a marking that is about to be checked, the concept of partial marking is introduced for 1-safe Petri nets in [132].

Definition 7.2 (Partial Marking)**.** A *partial marking* M_{par} of a 1-safe net system is a mapping $M_{par} : (P^1_{par} \cup P^0_{par}) \mapsto \{0,1\}$, where $P^1_{par}, P^0_{par} \subseteq P$ and $\forall p \in P^1_{par} : M_{par}(p) = 1$ and $\forall p \in P^0_{par} : M_{par}(p) = 0$. A partial marking is identified with the tuple $P'_{par} = (P^1_{par}, P^0_{par})$.

On the basis of the concept of partial marking, the authors of [132] gave a description for the reachability problem for 1-safe Petri nets (called REACHABILITY PROBLEM II in this work) as follows.

Definition 7.3 (REACHABILITY PROBLEM II)**.** Given a 1-safe Petri net system $\Sigma = (P, T, F, M_0)$ and a partial marking $P'_{par} = (P^1_{par}, P^0_{par})$, determine whether there is a marking M reachable from M_0 (i.e. $\exists \sigma : M_0 \xrightarrow{\sigma} M$) such that $\forall p \in (P^1_{par} \cup P^0_{par}) : M(p) = M_{par}(p)$.

When we determine whether there is a marking reachable from the initial marking such that a set of places are marked without considering places that should not carry a token, then REACHABILITY PROBLEM II can be simplified as REACHABILITY PROBLEM I.

Definition 7.4 (REACHABILITY PROBLEM I)**.** Given a 1-safe Petri net system $\Sigma = (P, T, F, M_0)$ and a partial marking $P'_{par} = (P^1_{par}, \emptyset)$, determine whether there is a marking M reachable from M_0 (i.e. $\exists \sigma : M_0 \xrightarrow{\sigma} M$) such that $\forall p \in P^1_{par} : M(p) = M_{par}(p)$.

In [139], REACHABILITY PROBLEM I is also expressed without using the concept of partial marking as follows.

Definition 7.5 (REACHABILITY PROBLEM I)**.** Given a 1-safe Petri net system $\Sigma = (P, T, F, M_0)$ and a subset $P' \subseteq P$, determine whether there is a marking M reachable from M_0 (i.e. $\exists \sigma : M_0 \xrightarrow{\sigma} M$) such that $\forall p \in P', M(p) = 1$.

As discussed previously, a configuration C of the complete prefix β_Σ of the unfolding of a 1-safe Petri net system Σ is associated with a unique reachable marking, denoted by $Mark(C)$, which define a set of places such that $\forall p \in Mark(C), M(p) = 1$ and $\forall p \notin Mark(C), M(p) = 0$. Thus the analysis for REACHABILITY PROBLEM II can be reduced to solve REACHABILITY PROBLEM I based on [139] as follows.

Proposition 7.6. *Let β_Σ be a complete prefix of the unfolding of a Petri net system $\Sigma = (P, T, F, M_0)$. A marking M is reachable in Σ iff β_Σ contains a configuration C such that $Mark(C) = M$.*

Proof. Suppose that $\beta_\Sigma = (ON, h)$ contains a configuration C, then $Mark(C) = h(Cut(C)) = h((Min(ON) \cup C^\bullet) \setminus^\bullet C)$ is a reachable marking (according to the definitions of configuration and cut). If $Mark(C) = M$, then the marking M is reachable in Σ. If a marking M is reachable in Σ, then there exists a firing sequence that leads the marking from M_0 to M. The mapping of this firing sequence to the complete prefix of the unfolding of the Petri net system β_Σ is a configuration C such that $Mark(C) = M$ (according to the construction process of β_Σ and the definition of configuration). □

Corollary 7.7. *Let $\Sigma = (P, T, F, M_0)$ be a 1-safe Petri net system, $P'_{par} = (P^1_{par}, \emptyset)$ a partial marking, and β_Σ a complete prefix of the unfolding of Σ. Then the partial marking P'_{par} is reachable in Σ iff β_Σ contains a configuration C such that $Mark(C) = P'_{par}$, and each linearisation of the configuration C is a solution.*

Proof. This corollary directly follows Proposition 7.6 and the definition of linearisation of configurations. □

7.2.3.2 On-the-fly Verification

To solve the reachability problems, *On-the-fly* verification was first suggested by McMillan [117] and further discussed e.g., in [132] and [139]. A new transition t_R is added to the original Petri net, whose preset is the set of places that are (or are not) wished to be marked simultaneously, i.e. $^\bullet t_R = \{p_1, \cdots, p_n\}$, and then construct the complete prefix of the unfolding of the new net. If the complete prefix contains any instance of this new transition, i.e. $h(e) = t_R$, the given marking is reachable, otherwise not.

Proposition 7.8 (On-the-fly Verification)**.** *Given a 1-safe Petri net system $\Sigma = (P, T, F, M_0)$, and a partial marking $P'_{par} = (P^1_{par}, \emptyset)$. Let $\Sigma_R = (P, T \cup t_R, F \bigcup_{p \in P'_{par}} (p, t_R) \cup (t_R, p), M_0)$, then:*

- *P'_{par} is reachable in Σ;*
- *$\Leftrightarrow$ there exists an event e in the complete prefix of the unfolding of Σ_R such that $h(e) = t_R$.*

Proof. If there exists an event e in the complete prefix of the unfolding of Σ_R such that $h(e) = t_R$, then there exists a marking M reachable from M_0 in the Petri net system Σ such that t_R is enabled. So the set of places $^\bullet t_R = P^1_{par}$ are able to be marked simultaneously which means P'_{par} is reachable in Σ. Suppose that P'_{par} is reachable in Σ, then the transition t_R such

that ${}^\bullet t_R = P^1_{par}$ is enabled under this marking. According to the construction process of the complete prefix of the unfolding of a Petri net system, an event corresponding to a transition that is enabled under a reachable marking can always be found in the the complete prefix of the unfolding. Therefore, there exists an event e in the complete prefix of the unfolding of Σ_R such that $h(e) = t_R$. □

7.2.4 Reachability Analysis for Coloured Petri Nets

7.2.4.1 Reachability Problems

Based on the presented knowledges of unfoldings of CP-nets, it is possible to extend the theory of reachability analysis for 1-safe Petri nets to Coloured Petri Nets. For this purpose, the concept of partial marking and reachability problem for CP-nets are defined.

For CP-nets, a marking is a function M that maps each place $p \in P$ into a multiset of tokens $M(p) \in C(p)_{MS}$. $M(p)(c)$ denotes the number of appearance of colour c in the multiset $M(p)$, i.e. the number of c-tokens on the place p.

Definition 7.9 (Partial Marking)**.** A partial marking $M_{\Omega_{par}}$ of a non-hierarchical CP-net $\Omega = (P, T, A, \Sigma, V, C, G, E, I)$ is a mapping $M_{\Omega_{par}} : (P^1_{par} \cup P^0_{par}) \mapsto \{M(p) \in C(p)_{MS}, \emptyset_{MS}\}$, where $P^1_{par}, P^0_{par} \subseteq P$ and $\forall p \in P^1_{par} : M_{\Omega_{par}}(p) = M(p) \in C(p)_{MS}$ and $\forall p \in P^0_{par} : M_{\Omega_{par}}(p) = \emptyset_{MS}$. A partial marking is identified with the tuple $P'_{par} = (P^1_{par}, P^0_{par})$.

Definition 7.10 (REACHABILITY PROBLEM II)**.** Given a non-hierarchical CP-net $\Omega = (P, T, A, \Sigma, V, C, G, E, I)$ and a partial marking $P'_{par} = (P^1_{par}, P^0_{par})$, determine whether there is a marking M reachable from M_0[7] (i.e. $\exists \sigma : M_0 \xrightarrow{\sigma} M$) such that $\forall p \in (P^1_{par} \cup P^0_{par}) : M(p) = M_{\Omega_{par}}(p)$.

Using the unfolding based techniques to solve reachability problems, as discussed, REACHABILITY PROBLEM II can be simplified as REACHABILITY PROBLEM I.

Definition 7.11 (REACHABILITY PROBLEM I)**.** Given a non-hierarchical CP-net $\Omega = (P, T, A, \Sigma, V, C, G, E, I)$ and a partial marking $P'_{par} = (P^1_{par}, \emptyset)$, determine whether there is a marking M reachable from M_0 (i.e. $\exists \sigma : M_0 \xrightarrow{\sigma} M$) such that $\forall p \in P^1_{par} : M(p) = M_{\Omega_{par}}(p)$.

The marking of a place of a CP-net is a multiset of tokens $M(p)$ and each token element (p, c) is mapped to a condition in the occurrence net, the number of conditions in the occurrence net that labelled with (p, c) is equal to $M(p)(c)$. Therefore the following conclusion can be drawn.

[7] The initial marking M_0 is defined by $M_0(p) = I(p)\langle\rangle$ for all $p \in P$.

Proposition 7.12. *Let $\Omega = (P, T, A, \Sigma, V, C, G, E, I)$ be a non-hierarchical CP-net, $P'_{par} = (P^1_{par}, \emptyset)$ a partial marking, and $\beta_\Omega = ((B, E, F), h)$ a complete prefix of the unfolding of Ω. Then the partial marking P'_{par} is reachable in Ω iff $\forall p \in P'_{par}, M(p)(c) = n$ where $c \in C(p)_{MS}$, β_Ω contains a configuration Con such that $b_1, \cdots b_n \in B, h(b_1) = \cdots = h(b_n) \in Mark(Con)$[8], and each linearisation of the configuration Con is a solution.*

Proof. See the proof of Corollary 7.7. □

7.2.4.2 On-the-fly Verification

Proposition 7.13 (On-the-fly Verification)**.** *Given a non-hierarchical CP-net $\Omega = (P, T, A, \Sigma, V, C, G, E, I)$, and a partial marking $P'_{par} = (P^1_{par}, \emptyset)$. Let $\Omega_R = (P, T \cup t_R, A \bigcup_{p \in P'_{par}} (p, t_R) \cup (t_R, p), \Sigma, V \cup Var(t_R), C, G \cup G(t_R), E \bigcup_{p \in P'_{par}} E(p, t_R) \cup E(t_R, p), I)$, then:*

- *P'_{par} is reachable in Ω;*
- *$\Leftrightarrow$ there exists an event e in the complete prefix of the unfolding of Ω_R such that $h(e) = t_R$.*

Proof. See the proof of Proposition 7.8. □

Remark 7.14. The reachability analysis for hierarchical CP-nets is similar to the analysis for the non-hierarchical ones except that whether a hierarchical or non-hierarchical CP-net is needed to be unfolded.

7.3 Application Example

There are two main subsystems in the SatZB: on-board subsystem and traffic control centre. Both CPN models of these two subsystems are hierarchically modelled. For reachability properties, analysing the entire system of SatZB model is too complicate because of the massive concurrent events in both subsystems. Besides, as a distributed system, the interactions between different subsystems of SatZB are of interest on the viewpoint of train control operation. Hence the reachability analysis for the entire system can be done by investigating the reachability properties of each subsystem model with respect to the standard interfaces.

In this section, we present examples how to verify the following verification tasks identified in chapter 4 by reachability analysis based on net unfoldings: *(1)the on-board module can switch*

[8] Con is used here to represent a configuration instead of C, since C represent the colour set of a CP-net in this section.

its activated scenario net to the net `OB_SN_Emergency_Stop` *from any other nets (except for the nets* `OB_SN_Initialisation`, `OB_SN_Registration` *and* `OB_SN_Logout`*); (2)the on-board module can not activate two or more than two scenario nets at the same time.*

7.3.1 Initialisation for the On-board Module

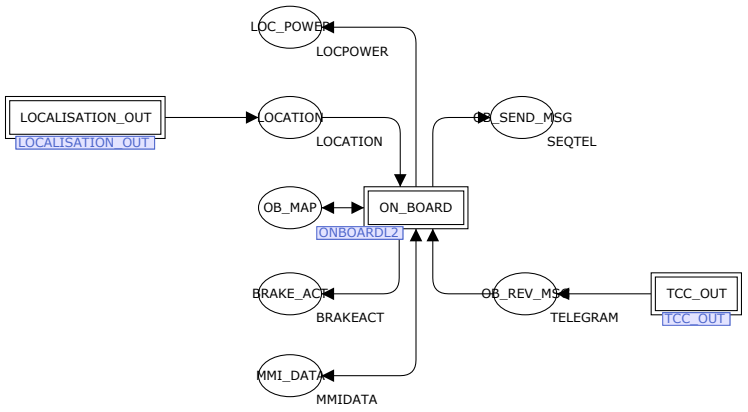
Recall the on-board module developed in chapter 3 (Figure 3.13). Places `LOCATION` and `OB_REV_MSG` are input channels and the markings (tokens) of these places represent messages that can be received by the on-board module but not yet received. Places `OB_SEND_MSG`, `LOC_POWER` and `BRAKE_ACT` are output channels and the markings of these places represent messages or signals that are about to be sent out by the on-board module but not yet sent out. Places `OB_MAP` and `MMI_DATA` are input/output channels that can both receive and send data. Nevertheless, place `OB_MAP` in this case is used as a resource and whose marking will never be changed, and place `MMI_DATA` is used only to deliver display data to the driver without considering getting input data from the driver through it.

The reachability properties of a system depends on its initial state and a component of a system is initialised by putting data on its input channels, therefore the input channels (places) of the on-board module should be initialised (marked). The input of the on-board module comes from the modules `LOCALISATION_UNIT` and `CENTRE` in the SatZB model. For the purpose of generating input data for the on-board module, a module represented by the substitution transition `LOCALISATION_OUT` and a module represented by the substitution transition `TCC_OUT` are added to the input channels (see Figure 7.9. The submodule `LOCALISATION_OUT`, shown in Figure 7.10, generates the possible location data for the on-board module. The submodule `TCC_OUT`, shown in Figure 7.11, produces all possible commands (messages) that are sent to the on-board module. Place `OB_MAP` is marked with a multiset representing the route map.

To verify the first verification task (e.g., the on-board module can switch its activated scenario net to the net `OB_SN_Emergency_Stop` from the net `OB_SN_Running`), we can assume that

1. only one location data ($(\text{“}29, UP\text{”})$) and one command ($\{TRAINID = 123, MSGID = \text{“}mt4\text{”}, TSTAMP = 3, DATA = \text{“}EMERGENCY\ STOP\text{”}\}$) are possible for the on-board module and
2. the currently activated scenario net of the on-board module is `OB_SN_Running`.

In order to represent the assumption 2, the place `Run Flag` in Figure 3.13 is initialised by the multiset $1'true$, which implies that the scenario net `OB_SN_Running` is activated at



1`{sequence = 1, ID1=1,LAT1="3599679.5513",LONG1="5797738.8174",HEIGHT1="78.4",LAT2="3599702.1687",LONG2="5797719.8690",HEIGHT2="78.4",MILEAGE1="0",LENGTH12="29"}++
1`{sequence = 2, ID1=0,LAT1="3599702.1687",LONG1="5797719.8690",HEIGHT1="78.4",LAT2="3599734.9683",LONG2="5797687.2684",HEIGHT2="77.5",MILEAGE1="29",LENGTH12="46"}++
1`{sequence = 3, ID1=0,LAT1="3599734.9683",LONG1="5797687.2684",HEIGHT1="77.5",LAT2="3599766.9168",LONG2="5797659.6596",HEIGHT2="76.7",MILEAGE1="75",LENGTH12="42"}++
1`{sequence = 4, ID1=2,LAT1="3599766.9168",LONG1="5797659.6596",HEIGHT1="76.7",LAT2="3599797.6459",LONG2="5797634.4381",HEIGHT2="76.5",MILEAGE1="117",LENGTH12="39"}++
1`{sequence = 5, ID1=0,LAT1="3599797.6459",LONG1="5797634.4381",HEIGHT1="76.5",LAT2="3599824.5652",LONG2="5797610.9022",HEIGHT2="76.4",MILEAGE1="157",LENGTH12="35"}++
1`{sequence = 6, ID1=3,LAT1="3599824.5652",LONG1="5797610.9022",HEIGHT1="76.4",LAT2="3599858.0133",LONG2="5797582.3040",HEIGHT2="76.5",MILEAGE1="193",LENGTH12="44"}++
1`{sequence = 7, ID1=0,LAT1="3599858.0133",LONG1="5797582.3040",HEIGHT1="76.5",LAT2="3599887.8692",LONG2="5797555.8781",HEIGHT2="76.7",MILEAGE1="237",LENGTH12="39"}++
1`{sequence = 8, ID1=4,LAT1="3599887.8692",LONG1="5797555.8781",HEIGHT1="76.7",LAT2="3599920.0571",LONG2="5797530.5756",HEIGHT2="76.7",MILEAGE1="277",LENGTH12="40"}++
1`{sequence = 9, ID1=0,LAT1="3599920.0571",LONG1="5797530.5756",HEIGHT1="76.7",LAT2="3599946.0076",LONG2="5797507.7627",HEIGHT2="77.2",MILEAGE1="318",LENGTH12="34"}++
1`{sequence = 10, ID1=5,LAT1="3599946.0076",LONG1="5797507.7627",HEIGHT1="77.2",LAT2="3599946.0076",LONG2="5797507.7627",HEIGHT2="77.2",MILEAGE1="352",LENGTH12="0"}

FIGURE 7.9: The on-board module with input modules

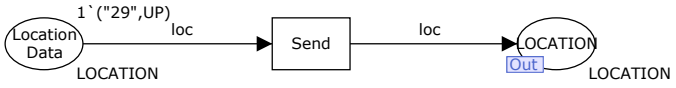


FIGURE 7.10: Submodule LOCALISATION_OUT

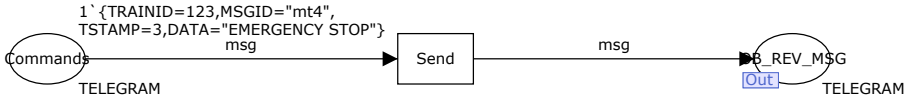


FIGURE 7.11: Submodule TCC_OUT

present. This means that data processing in the scenario nets OB_SN_Initialisation and OB_SN_Registration has been accomplished. Therefore, fusion places Section Points, MA Points and Logout Points in Figure 3.17 are marked with multisets that produced during the initialisation process on the net OB_SN_Initialisation. Places MA Request and Logout Request are initialised by constant multisets that are used to generate MA request and logout request, respectively.

7.3.2 Unfold the On-board Module

Apart from establishing CPN models, CPN Tools is also adequate for manually drawing the unfolding of a CPN model. Places represent conditions and transitions represent events of the unfolding. For convenience, a pair of a place's name and its marking indicates a token element. The name of a transition implies the net that it belongs to.

According to the procedures of direct unfolding for hierarchical CP-nets presented in this chapter, the unfolding of the on-board module with the given assumption is illustrated in

Figure 7.12. There are five token elements in the initial marking corresponding to conditions b_1, b_2, b_3, b_4 and b_5. Event e_4 is a cut-off event. Note that for saving space, (1) the markings of places `Section Points`, `MA Points`, `Logout Points`, `MA Request` and `Logout Request` are omitted because they are connected to the transition with double directed (test) arcs, which means these markings are invariant.(2) the transition `Add SeqNum` in Figure 3.13 is also omitted. (3) assuming that once a message is put on the output channels it will be sent out immediately, then at most one token could be existed on places `OB_SEND_MSG` and `MMI_DATA`.

Note that the transition `Add SeqNum` in Figure 3.13 is omitted for the reason of saving space for the unfolding. Since the markings of places `Section Points`, `MA Points`, `Logout Points`, `MA Request` and `Logout Request` are constant, these places can be omitted while investigate the reachability properties. Assuming that once a message is put on the output channels it will be sent out immediately, then at most one token could appear on places `OB_SEND_MSG` and `MMI_DATA`.

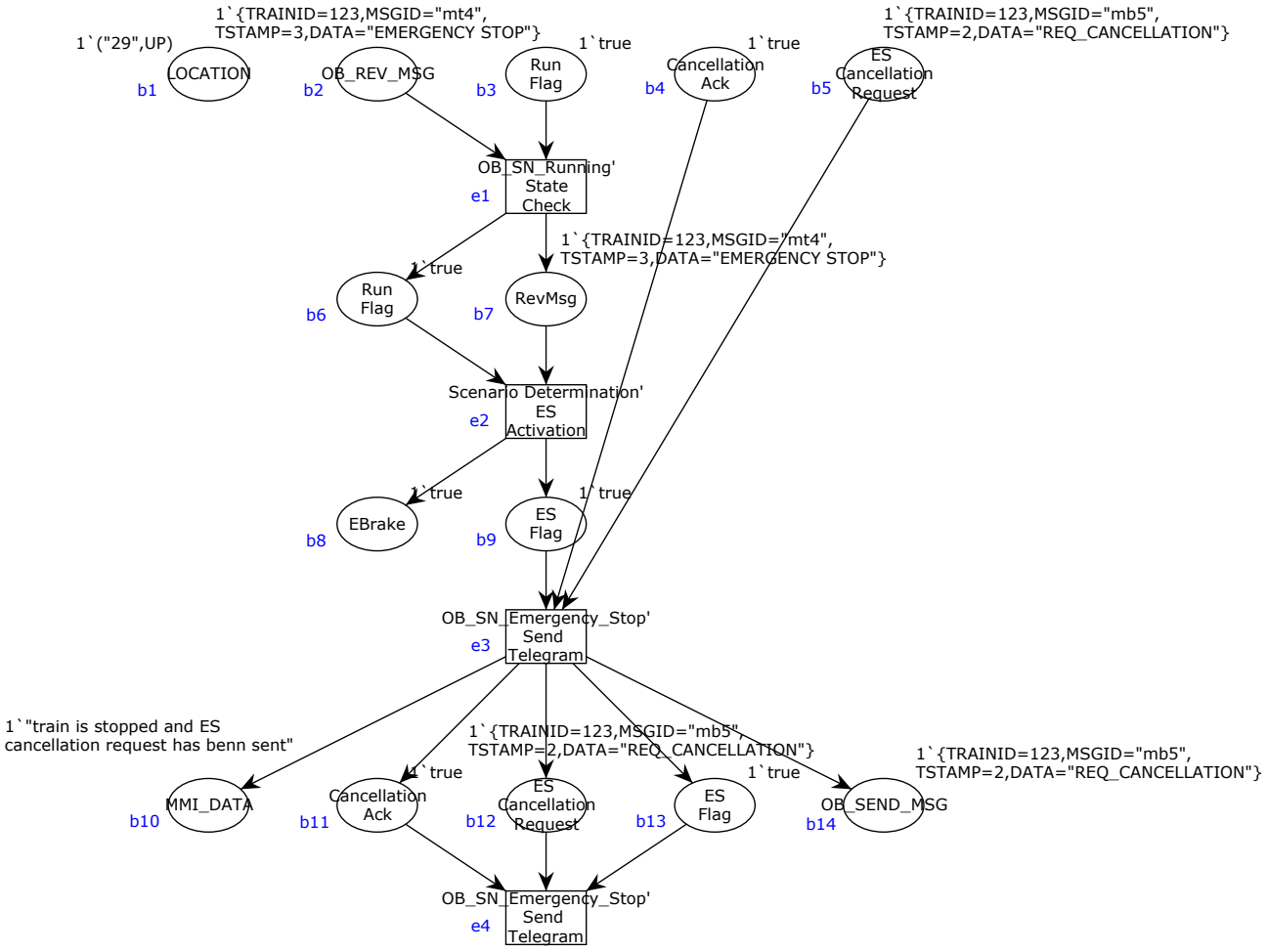


FIGURE 7.12: Finite complete prefix of the unfolding of the on-board module

7.3.3 Reachability Analysis for the On-board Module

The unfolding of a Petri net model encodes all reachable markings, so all reachable markings of the on-board module under the given initial marking can be investigated with Figure 7.12. A marking that can be reached from the initial marking of the on-board module is a mapping of the cut of a configuration.

Verification task: whether the on-board module can switch its activated scenario net to the net `OB_SN_Emergency_Stop` from the net `OB_SN_Running`.

Expected partial marking: $P'_{par} = ((ES\ Flag, 1`true), (EBrake, 1`true), (OB_SEND_MSG, \{TRAINID = 123, MSGID = \text{“}mb5\text{”}, TSTAMP = 2, DATA = \text{“}REQ_CANCELLATION\text{”}\}), (MMI_DATA, 1`train\ is\ stopped\ and\ ES\ cancellation\ request\ has\ been\ sent))$. It means that the activated scenario net has switched from `OB_SN_Running` to `OB_SN_Emergency_Stop`.

Analysis: this is a problem of REACHABILITY PROBLEM I. According to Proposition 7.12, it is easy to identify a configuration of the prefix $Con = \{e_1, e_2, e_3\}$ such that $Mark(Con) = P'_{par}$. Therefore, the partial marking P'_{par} is reachable from the given initial marking, which is expected. The linearisation $e_1\ e_2\ e_3$ is the solution for the reachability problem.

Analysis result: the on-board module can switch its activated scenario net to the net `OB_SN_Emergency_Stop` from the net `OB_SN_Running`.

Discussion: the verification of the on-board module that whether it can switch its activated scenario net to the net `OB_SN_Emergency_Stop` from other nets can be conducted similarly.

7.3.3.1 On-the-fly Verification of the On-board Module

The algorithm used in this thesis for constructing the (finite) complete prefix of the unfolding is proposed by McMillan [118]. Besides, there is an improved algorithm presented in [119] based on the total order semantics of Petri nets either than the partial order semantics of Petri nets that McMillan exploited. Whichever algorithm to be adopted, the generated complete prefix of the unfolding is much larger than necessary from time to time. Unfolding process could be stopped when the marking to be checked has already contain in the prefix of the unfolding. Therefore, the approach of on-the-fly verification is desirable.

Verification task: whether the on-board module can activate two scenario nets at the same time, for example, whether the activated scenario net(s) of the the on-board module can be `OB_SN_Running` and `OB_SN_Conditional_Running` simultaneously. For this purpose, the

initial marking on the place Command in Figure 7.11 needs to be replaced by $1`\{TRAINID = 123, MSGID = \text{“}mt3\text{”}, TSTAMP = 3, DATA = \text{“}CONDITIONAL\ RUNNING\text{”}\}$.

Unexpected partial marking: $P''_{par} = (((Run\ Flag, 1`true), (CondRun\ Flag, 1`true)), \emptyset)$. It represents that scenario nets OB_SN_Running and OB_SN_Conditional_Running of the on-baord module are activated simultaneously.

Analysis: according to the theory of on-the-fly verification presented in section 7.2.4.2, the transition New is added to the scenario net OB_SN_Running shown in Figure 7.13. The preset of the transition New are the places Run Flag and CondRun Flag. The unfolding of the modified on-board module could be shown as in Figure 7.14. Event e_4 is a cut-off event. It is observed that no event in the prefix is mapped to the transition New in the modified on-board module. Thus, the partial marking P''_{par} is not reachable from the given initial marking.

Analysis result: the on-board module can not activate the scenario nets OB_SN_Running and OB_SN_Conditional_Running at the same time.

Discussion: other situations (i.e., two or more than two other scenario nets are activated simultaneously) can be verified similarly. We can see that if there is no error in the original Petri net, then a complete prefix of the unfolding of the Petri net is required. In this case, the on-the-fly verification method is just as efficient as the verification with normal unfolding-based approach. However, if there are errors in the original Petri net, then verification using on-the-fly method could be much more efficient in some cases. For example, if the arc from the place Run Flag to the transition CondRun Activation of the submodule Scenario Determination in Figure 3.19 is specified with a double directed (test) arc as shown in Figure 7.15 by mistake, then a (incomplete) prefix of the unfolding of the on-board module depicted in Figure 7.16 is sufficient to check the partial marking P''_{par}. The event e_3 in the prefix is mapped to the transition New in the original module, so the partial marking P''_{par} is reachable. It is observed that the prefix in Figure 7.16 is smaller than the prefix in Figure 7.14. This kind of size difference of prefixes could be considerable in some extreme cases. Therefore, the on-the-fly verification approach is especially suitable for checking a marking that is likely to be reachable.

The state space of the modified on-board module with respect to Figure 7.15 and Figure 7.13 is infinite, because once the place CondRun Flag is marked, the transition Output Data in the scenario net OB_SN_Conditional_Running (see Figure 3.22) can fire infinitely. In this circumstance, only a partial sate space can be generated as shown in Figure 7.17, and it took hours to generate this report. It is thus infeasible to investigate reachability properties by using state space based techniques in this case.

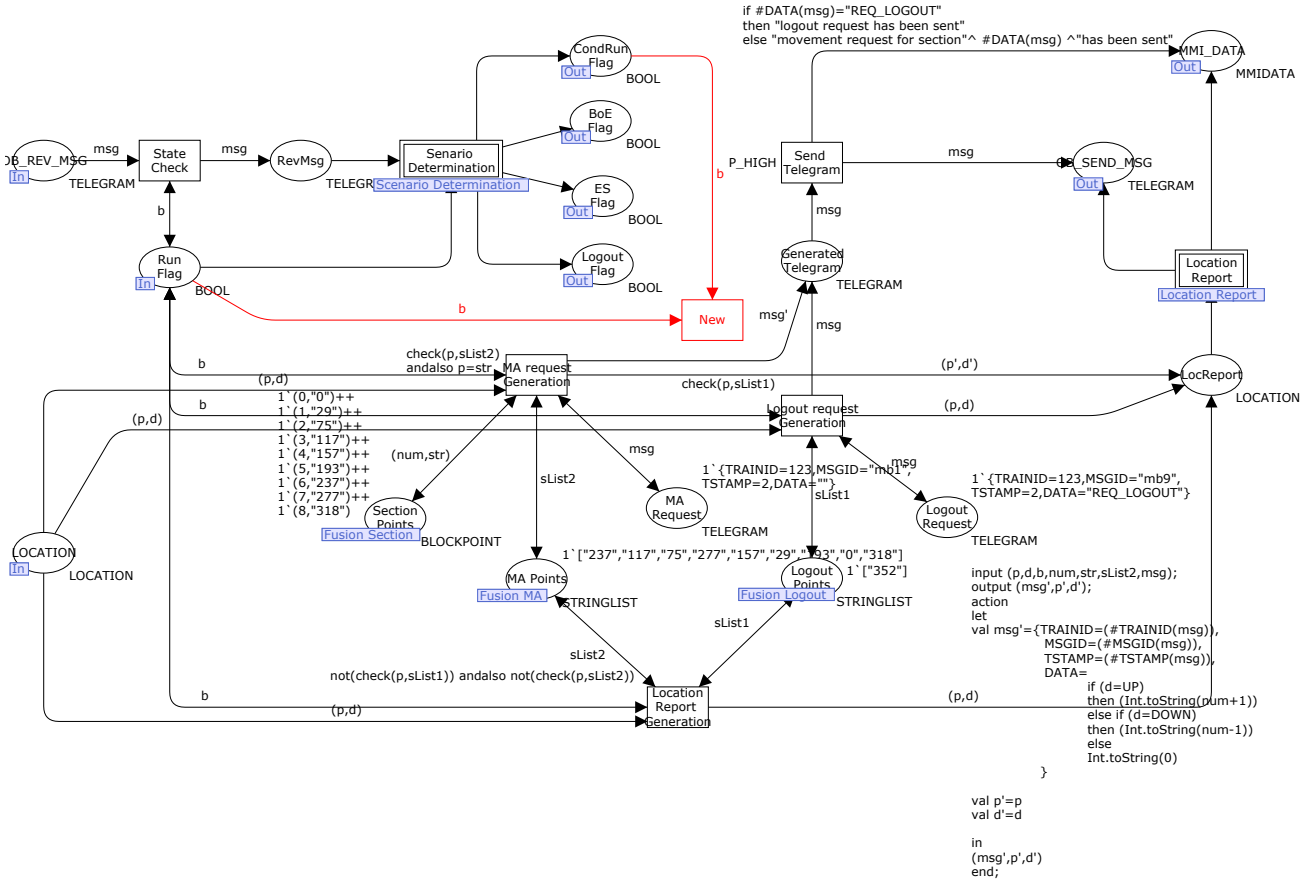


FIGURE 7.13: New scenario net `OB_SN_Running`

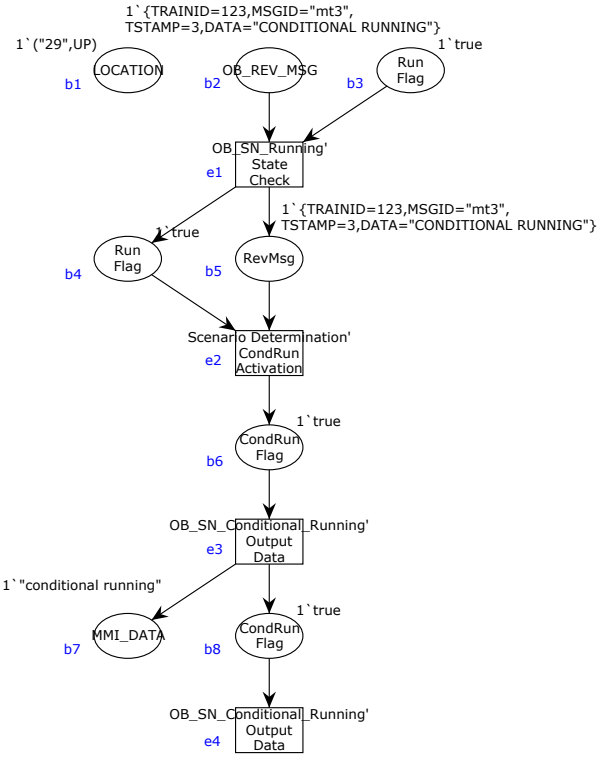


FIGURE 7.14: Finite complete prefix of the unfolding of the modified on-board module with respect to Figure 7.13

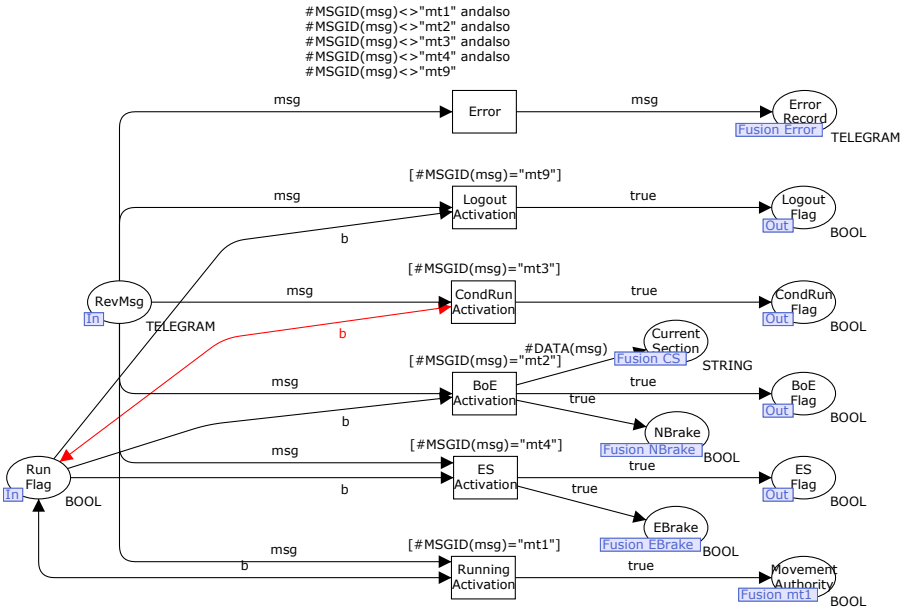


FIGURE 7.15: Unexpected change for Figure 3.19

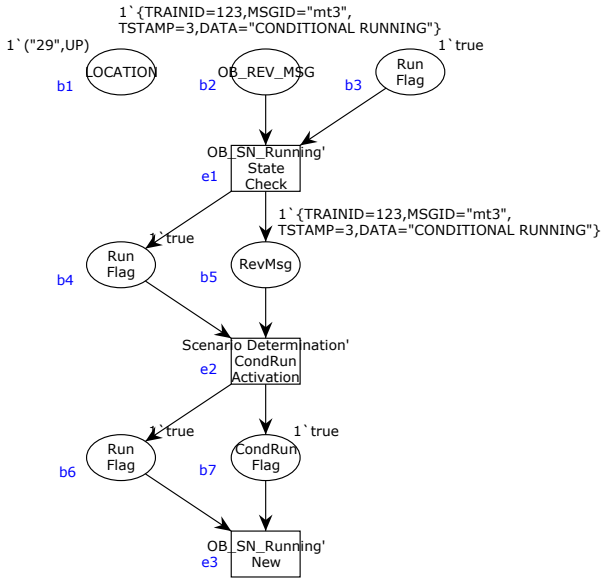


FIGURE 7.16: Prefix of the unfolding of the modified on-board module with respect to Figure 7.15 and Figure 7.13

7.3.4 Complexity of the Unfolding-based Approach

To investigate the reachability property of a component of a distributed system, the inputs of the component are considered as the initialisations of the component. If the input consists of more than one port and each port may be filled with different values, then a possible combination of the filled value of each port is an initialisation. The number of the finite complete prefix of the unfolding is equal to the number of possible initialisations of the component. Assume that the number of input port of a component is n and the number of possible value for each port is $m_1, m_2, \cdots, m_n$, then the number of the required finite complete prefixes of the

```
CPN Tools state space report for:
/cygdrive/F/Wu Daohua/Dissertation/Thesis_WD/Models
Report generated: Wed Mar 20 19:01:31 2013


 Statistics
------------------------------------------------

  State Space
     Nodes:  17019
     Arcs:   23820
     Secs:   300
     Status: Partial

  Scc Graph
     Nodes:  17019
     Arcs:   23820
     Secs:   18
```

FIGURE 7.17: Partial state space

unfolding is $m_1 \cdot m_2 \cdots m_n$. In the case of the on-board module of SatZB model, there are two input ports (places `LOCATION` and `OB_REV_MSG`). For the place `LOCATION`, there are 4 possible values (i.e., MA point, not MA point, point of entering, point of leaving). For the place `OB_REV_MSG`, 9 values (i.e., `mt1`,`mt2`,$\cdots$,`mt9`) are possible. So the number of the required finite complete prefixes of the unfolding is 36.

7.4 Summary

This chapter first gives a general introduction of the existing techniques for investigating reachability properties of Petri nets. And then we focus on the techniques based on Petri net unfoldings. After providing the notations and basic definitions of net unfoldings, the formalisation of unfolding hierarchical CP-nets based on the approach for non-hierarchical CP-nets proposed in [39] is presented. To describe a marking that is going to be checked in reachability analysis, the concept of partial marking is introduced. In general, reachability problems could be identified as REACHABILITY PROBLEM I or REACHABILITY PROBLEM II. However, REACHABILITY PROBLEM II can be substituted by REACHABILITY PROBLEM I when it comes to solving reachability problems by unfolding-based techniques. In particular, the on-the-fly verification approach is highlighted for checking a marking that is likely to be reachable. Otherwise, the complete prefix of the unfolding is required. The unfolding-based approach of investigating the reachability properties of 1-safe Petri nets are extended to CP-nets in this chapter. At last, some verification tasks identified in chapter 4 are verified by reachability analysis based on Petri net unfoldings.

Personal Contribution. In this chapter, unfolding hierarchical CP-nets based on the approach for non-hierarchical CP-nets proposed in [39] is introduced and the procedure of directly unfolding hierarchical CP-nets is formalised. Reachability problems [132], [139] for Petri nets are extended from 1-safe Petri nets to CP-nets, and distinguished between REACHABILITY PROBLEM I and REACHABILITY PROBLEM II although REACHABILITY PROBLEM II can be substituted by REACHABILITY PROBLEM I when one solves reachability problems with net unfoldings. The formal definitions of partial marking, REACHABILITY PROBLEM II and REACHABILITY PROBLEM I of non-hierarchical CP-nets as well as the propositions of reachability analysis for non-hierarchical CP-nets are presented. Reachability analysis based on net unfoldings is applied to the on-board module of a satellite-based train control system.

Chapter 8

Testing Based on Petri Nets

In this chapter, model-based testing are exploited for testing the the on-board module of SatZB model established in chapter 3 focusing on verifying the functionality. First, the terminology of testing is introduced. Second, the methodology applied in this chapter is presented. Third, two test generation techniques based on CPNs and SPENAT (Safe Place Transition Nets with Attributes) respectively are proposed to generate the test suite. Finally, a comparison of the two techniques and a test evaluation are carried out.

8.1 Testing

Testing denotes a set of activities that aim either at showing that actual and intended behaviours of a system differ, or at increasing confidence that they do not differ [140]. Testing plays an important role during the system development, as there should be a specific level of testing corresponds to each phase of the life-cycle of the system. This is known as the "V" model of testing, which is depicted in Figure 8.1. Generally, unit testing, integration testing, system testing and acceptance testing are performed by the programmer on the unit, the development team on the integrated system, the test group to verify system requirements, and a dedicated group (with customers) to verify the user requirements, respectively.

In the European standard EN50126 [22], the objectives of verification and validation are described as following: the objective of verification is to demonstrate that, for the specific inputs, the deliverables of each phase meet in all respects the requirements of that phase; the objective of validation is to demonstrate that the system under consideration, at any step of its development and after its installation, meets its requirements in all respects. In other words, validation is answering the question of "*are we building the right thing?*" and verification is

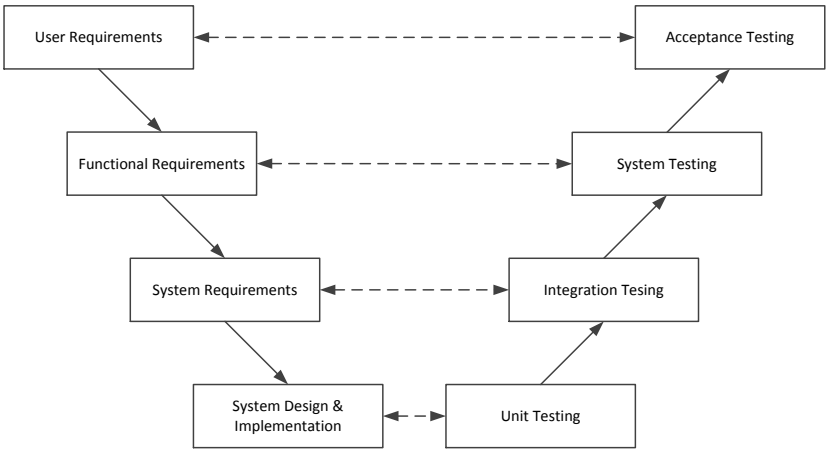


FIGURE 8.1: The "V" model of testing

answering the question of "*are we building it right?*" In this sense, testing is a verification as well as a validation technique.

8.1.1 Terminology

The term "test" has a very general meaning covering from unsystematic, not quantifiable and not reproducible implementations to automatic and systematic testing. Depending on the complexity of the system under test, the significance of the test terminology is variant. For example, to test a door whether it can open and close as normal, the requirements and the expected output are quite clear and simple, so we may implement the test without considering the terminology. However, if the system under test is a complex system such as a train control system whose requirements specification is usually relatively abstract (usually in informal texts), further analysis of the specification down to the quantification of the test purpose is needed. Besides, testing a complex system normally involves many persons or teams, e.g., system design and development team on one side and test team on the other side. Moreover, the persons involved may have different education or training background, which makes an consistent and error-free communication between these persons vital for the testing. Therefore, defining the terminology of testing has a significant importance to the testing work. In this section, the terms related to this work are defined.

Test process: a *test process* consists of the activities which are needed for testing a *test object*. On the basis of [41] and [141], a test process can include the sequential activities of *test planning*, *test specification*, *test construction*, *test execution* and *test evaluation*. Each of these activities is called a *test phase*.

Test planning: the activity of establishing or updating a test plan, which is a document describing the scope, approach, resources and schedule of intended test activities [142]. In particular, the *test strategy* and *test purpose* are defined under the consideration of realising the test process with affordable cost.

Test strategy: the *test strategy* specifies the abstraction levels where tests will be executed and the different systems under test, e.g. units or the complete system [143]. In other words, it specifies the *test object* and *test level*.

Test object: a *test object* is an object which is under test. This object can be a system or an implementation, and is called *system under test* (SUT) and *implementation under test* (IUT), respectively. Note that the IUT is usually distinguished from the SUT. An IUT is the implementation that one wants to test. An SUT is the IUT together with its *test context*, elements one does not want to test, but are needed to access the IUT [144]. In this chapter, the on-board module of SatZB model is considered as the test object (i.e., the IUT).

Test context: a *test context* consists of things that one does not want to test, but are "in the way" between the tester and the thing that one wants to test [144]. In this chapter, the module of localisation unit together with the TCC module is the test context of the on-board module.

Test level: the *test level* is with correspond to the abstraction level of the testing [141], e.g., unit testing, integration testing, system testing and acceptance testing in the "V" model of testing shown in Figure 8.1. In this chapter, testing for the on-board module could be seen as a unit testing.

Test purpose: a *test purpose* is a property one wants to test [144]. For example, "conformance", "statement coverage", "an expected coverage degree of specification" [144], [41]. These properties can be expressed informally or formally, but in general, they can not be used for testing a system directly because they specify possibly infinite runs of a system or it is not clear how to derive test cases. In [145], demonstration, detection, and prevention are identified as three important test purposes. Demonstration shows the confidence in system functioning. Detection finds the defects, errors and system deficiencies of a system. Prevention as a means of quality management has a magnificent meaning in system development. It can prevent or reduce the likelihood of errors being made from the very beginning of a system development. In this chapter, the test purpose of testing the on-board module is to fully cover the functional requirements specification so as to ensure the quality of the module and show the confidence in system functioning.

Test specification: the activity of specifying the *test selection criterion* and *test case specification* [141].

Test selection/coverage criteria: a *test selection/coverage criterion* defines a method for the selection of test cases (path coverage, state coverage, function coverage, etc.) and a measure for the definition of the size of the test [146]. In this chapter, the test selection criterion is "all state sequences" criterion (see chapter 5) relates to the behaviour model of the SUT.

Test case specification: the *test case specifications* formalise the notion of test selection criteria and render them operational [147]. For instance, a test selection criterion "state coverage" would be translated into a test case specification in the form of "reach θ" for all states θ of the finite state space, concerning possibly further constrains on the length and the number of the test cases. In [144], a test case specification formally represents a *test suite*. It must be well-defined how to derive a test suite from a test case specification. Hence, it requires knowledge of the intended behaviour (specification or behaviour model) of the SUT. If an explicit behaviour model (called *test model* in model-based testing) exists, then the test case specification can be understood as a selection criterion on the set of model traces. In this chapter, the test case specification translates the "all state sequences" criterion into "all paths of the reachability tree" (see chapter 5) with respect to the the behaviour model of the SUT.

Test oracle/reference: the term *test oracle* is usually used in software testing and software engineering to determine whether a test has passed or failed. According to [148], a test oracle is a mechanism using to generate expected results for each *test case* in order to check the test results. In [149], a complete oracle have three capabilities and carry them out perfectly:

- a generator, to provide predicted or expected results for each test.
- a comparator, to compare expected and obtained results.
- an evaluator, to determine whether the comparison results are sufficiently close to be a pass.

In [141], the term *test reference* used to derive test input before test execution and comparing test results with the expected results, and therefore has a similar character as test oracle. Test reference or test oracle could be in the form of concept in mind, specification, model and so on [141]. In correlation with model-based testing, the term *test model* is also used, although it is restricted to model-based testing. In this chapter, the test oracle/reference refers to the test model.

Test case: a *test case* is a specification which consists of input and expected output. On the basis of test reference and test specification, test cases can be derived. The input part of a test case is call *test data* [144]. In general, test cases will also include additional information such as descriptions of execution conditions or applicable configurations. Besides, the term *test*

suite is used widely in testing techniques. A test suite is a finite set of test cases [147]. In this chapter, a test case is a pair consisting of an initial state and a finite firing sequence (message sequence).

Test construction: the activity of integrating the test object, test context, and *test adaptor*.

Test adaptor: a *test adaptor* is a program using to code the test data of a test case for the SUT, and decode the test output of a test execution. The term *test driver* also used as test adaptor in [41]. In this chapter, the adaptation work such as transforming the message "mt1" into "{TRAINID=123,MSGID='mt1',TSTAMP=3,DATA='PERMIT RUNNING'}" is done by hand.

Test execution: the activity of applying test data to the SUT, monitoring the behaviour of the SUT, and comparing expected and actual behaviours in order to yield a *verdict*.

Verdict: a *verdict* is the result of the comparison of the (actual) output of the SUT with the expected provided by the test case [144]. In general, a verdict could be pass (the actual output conforms to the expected output), fail (the actual output does not conform to the expected output) or inconclusive (the conclusion of conformance cannot be driven).

Test evaluation: the activity of evaluating the tests for requirements coverage and test completeness [150]. The evaluation of the requirements coverage could be done by assessing the extent of the SUT exercised, while the test completeness is able to implemented by determining whether the set of inputs used during the test are a fair representative sample from the set of all possible inputs to the SUT.

In order to have a overview of the process of testing and related terms presented above, Figure 8.2 is provided.

8.1.2 Model-based Testing

Normally, testing is to verify the actual behaviour in conformance with the intended behaviour of the SUT. The intended behaviour of a system usually relies on the requirements specification documents, from which a test model for generating test case automatically can be derived. The idea of model-based testing is to use explicit behaviour models to encode the intended behaviour [140]. Traces are generated from the model according to the test case specifications. The traces of these models are interpreted as test cases for the SUT.

Model-based testing is using a model to generate test cases, and then running the test cases on the SUT. Note that in model-based system development, the SUT is valid to be substituted by models. The process of model-based testing can be depicted in Figure 8.3.

	Step	Test phase	Input	Output		Example
Test process	1	Test planning	Requirements specification	Test strategy	Test object (IUT)	on-board module
					Test level	unit testing
				Test purpose		cover the full functional requirements specification so as to ensure the quality of the on-board module and show the confidence in system functioning
	2	Test specification	Test strategy, Test purpose	Test selection/coverage criteria		all state sequences criterion
				Test case specification		all path of the reachability tree
				Test oracle/reference		test model
			Test selection/coverage criteria, Test case specification, Test reference	Test cases		a set pairs consisting of an initial sate and a finite firing sequence (message sequence)
	3	Test construction	Test cases	Test adaptor		transform the message “mt1” into “{TRAINID=123,MSGID=‘mt1’, TSTAMP=3,DATA=‘PERMIT RUNNING’}”
			Requirements specification	Test context		models of localisation unit and TCC
	4	Test execution	Test adaptor, Test context, Test cases	Verdicts		tests are passed, failed or inconclusive
	5	Test evaluation	Verdicts	Requirements coverage and test completeness		all requirements are covered and the test is complete

FIGURE 8.2: Terminology of testing

8.2 Methodology

Aiming at developing a satellite-based train control system with the model-based and tool-supported automated code generation approach, we propose two model-based test generation techniques for verifying the on-board module of SatZB model (test object). By applying two or even more (model-based) test generation techniques, *systematic errors* of test generation could be avoided. If the test cases generated by two different test generation techniques are essentially the same, then we have the confidence saying that the test models are correct and the test cases are adequate. Figure 8.4 illustrates the methodology employed in this chapter. The gray blocks “Model-based Test Generation” in Figure 8.4 is the main task. Based on the system requirements specification, two test models, i.e., a CPN model and a SPENAT model, respectively are developed in the first place. For the CPN model a reachability graph is generated by state space analysis, while for the SPENAT model the complete prefix

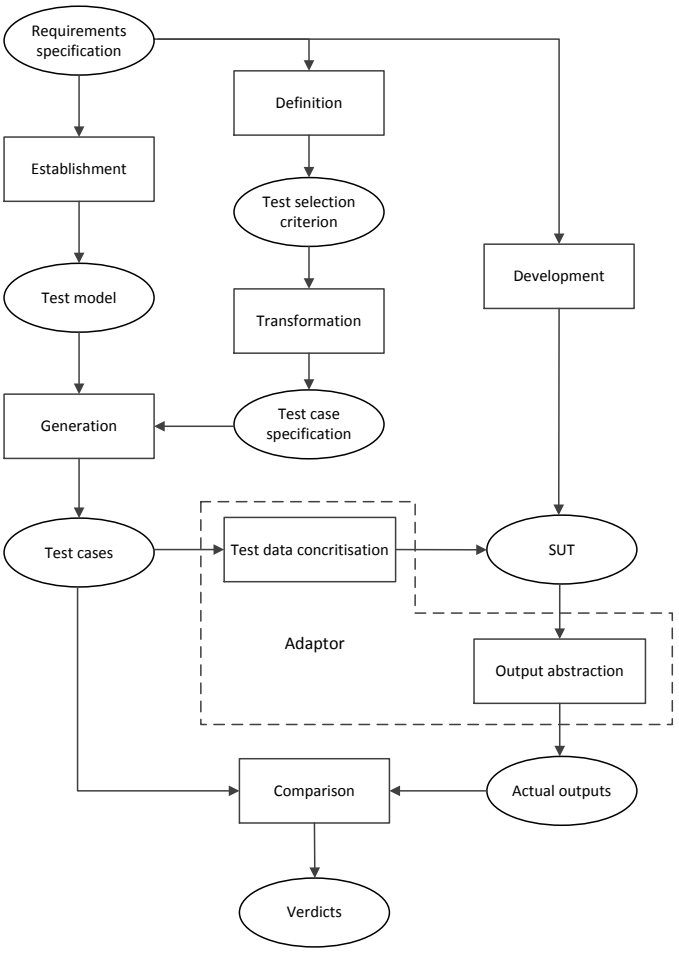


FIGURE 8.3: Model-based testing process

of the SPENAT unfolding is constructed with net unfolding techniques. Test cases are then derived either from the reachability graph using the *RT-based method* (see chapter 5) or from the complete prefix of the SPENAT unfolding applying the *unfolding-based method* (see chapter 5) with the "all state sequences" test coverage criterion. These test cases are occurrence sequences which can be represented by messages sequences. When they are applied to the system model as stimuli, the verification and/or validation of the system model by testing could be implemented.

8.3 Test Generation Based on CPNs

Petri nets are well-known as adequate means of descriptions for distributed systems and the system behaviour of concurrency. The environment of the on-board subsystem of SatZB, i.e., the traffic control centre, is characterised by a concurrency of delivering different commands

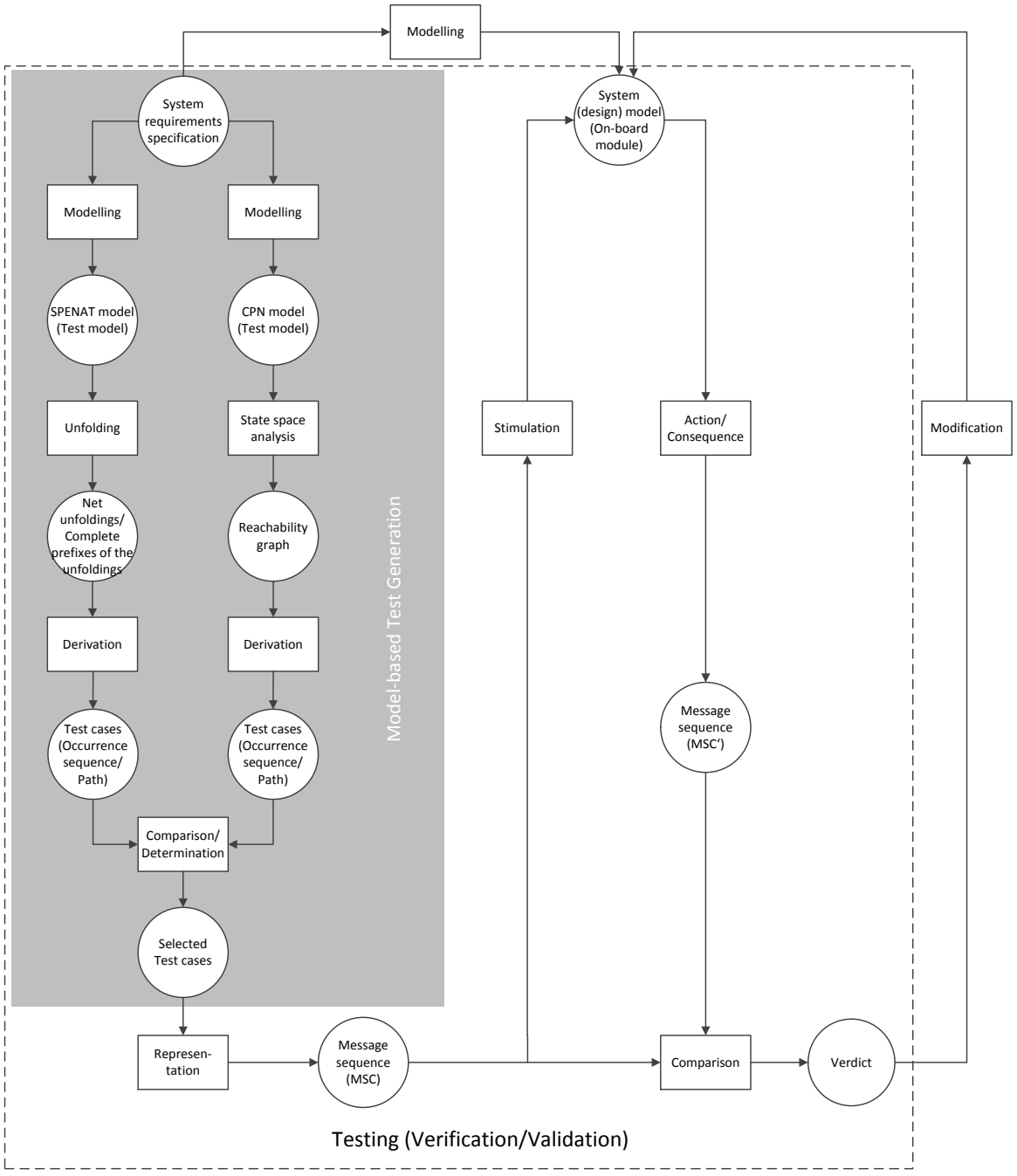


FIGURE 8.4: Methodology of the model-based testing for the design model

to the on-board subsystem while taking the on-board subsystem as the test object. Thus (Coloured) Petri nets are feasible to be used for developing the test model.

8.3.1 Test Model

The CPN modelling language supports the specification of hierarchically structured models, which makes it possible to work with different levels of detail and abstraction.

For system development, the design model is dedicated to the structure and construction of the system. On the contrary, the test model concentrates the behaviours that are of interest. The test model is more abstract or "simpler" than the design model since the test model is usually focus on a specific aspect of the system, by doing which fewer test cases are required. In this work, the behaviour of scenario net transitions of the on-board module of SatZB model is under consideration. This is because covering all sequences of scenario net transitions indicates that all scenarios (e.g. Figure 3.6) and possible scenario sequences specified in the functional requirements specification are covered according to the model construction mechanism applied in chapter 3. The transitions of scenario nets of the on-board module has been illustrated in Figure 3.14. The modules of the test model that represent the operating environment of the on-board subsystem model encode the behaviours of the modules `CENTRE` and `LOCALISATION_UNIT` in Figure 3.12.

8.3.1.1 Top Level

The top level of the test model is shown in Figure 8.5. Modules `LocUnit` and `TCC` model the environment of the test object. Places `Location Data`, `Output Message` and `Input Message` represent the communication channels between different modules. These communication channels are simplified as one-way channels under the condition that the train control system is a real-time system, and only one message would be transmitted on a communication channel at the same time. Moreover, we suppose that the communication system between the on-board subsystem and the traffic control centre is fully reliable.

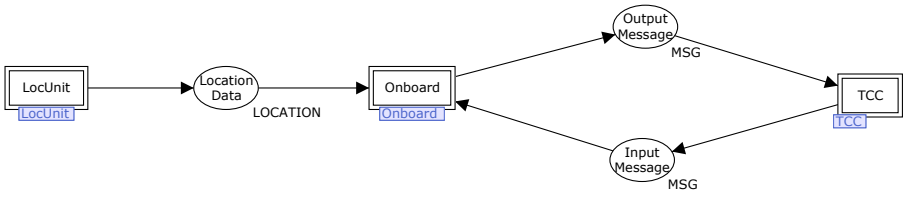


FIGURE 8.5: Top level of the test model

8.3.1.2 Second Level

In this level, the behaviour of scenario net transitions illustrated in Figure 3.14 are described.

Submodule `Onboard`. The main idea of this page is to reflect the interactions between the on-board subsystem model and its environment illustrated in the sequence diagrams. From the given interpretation of the sequence diagram in Figure 3.6, we conclude that sending movement requests and switchovers of scenario nets of the on-board subsystem model are determined by the received location data from the model of the localisation unit and the messages from the model of traffic control centre. This is depicted in Figure 8.6. When a location data is received from the place `Location Data`, it is compared to the on-board map (tokens on the place `OB_Map MA Points`), and a specific message (either `mb1` or `mb5`, see Table 3.2) might be sent out via the place `Output Message` under the consideration of the marking on the place `Onboard Scenario`. The currently activated scenario net of the on-board subsystem model, indicated by the marking on the place `Onboard Scenario`, is varied depending on the messages received from the place `Input Message`. Place `Serial Number` is used to synchronise the submodules `Onboard` and `LocUnit`, which will be introduced in the following subsections.

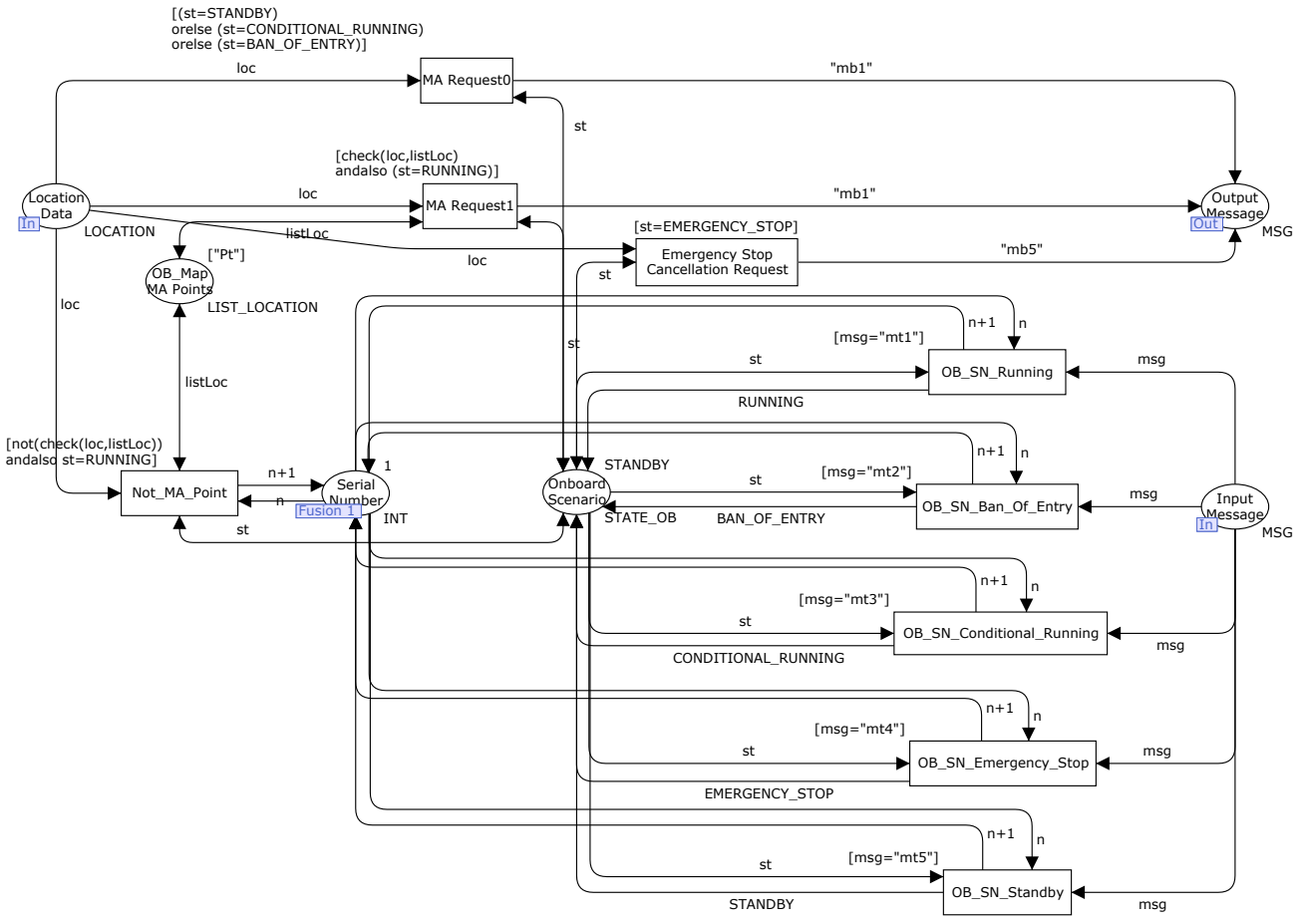


FIGURE 8.6: Page of the submodule `Onboard`

Submodule `LocUnit`. The location data are the stimuli that trigger the operation of the designed system model, so the submodule `LocUnit` focus on putting out location data to the place `Location Data` in a desired order and the time interval between two consecutive data. The order of the output data is controlled by adding an additional indicator (integer number) on each token. The time interval between two consecutive data is controlled by using the fusion place `Serial Number`. According to Figure 8.6 and Figure 8.7, we can see that only the received message from the module `TCC` has been processed, the integer number on the place `Serial Number` can be increased by one. Then the next location data can be put out.

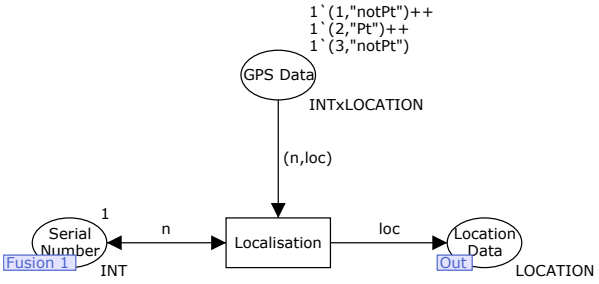


FIGURE 8.7: Page of the submodule `LocUnit`

Submodule `TCC`. The task of the submodule `TCC` is to send commands (e.g., movement authorities) to the submodule `Onboard` as the model of the traffic control centre does in the designed system model, but only considering all possible commands in a simple way. In Figure 8.8, when a movement request `mb1` is received by the place `Output Message`, one of the following four commands `mt1`, `mt2`, `mt3`, or `mt4` (see Table 3.3) could be sent out as a response via the place `Input Message`, and which one to be sent is random; if the cancellation of emergency stop request `mb5` is received, then either `mt4` or `mt5` will be sent out. It describes a situation that if more than one command could be sent out (to the on-board subsystem), then all the possible commands have the same probability to be delivered.

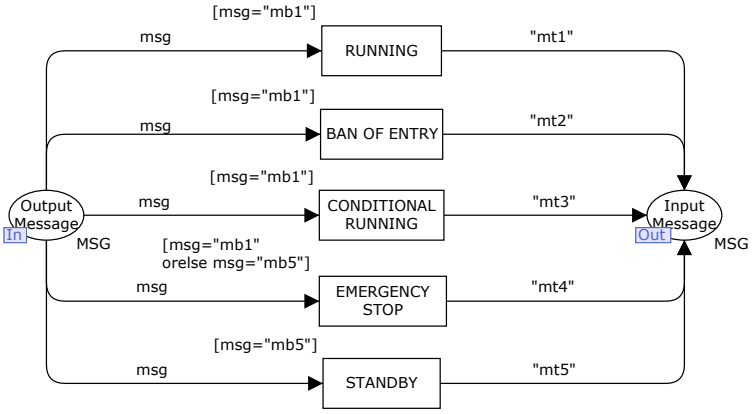


FIGURE 8.8: Page of the submodule `TCC`

8.3.2 Test Suite Generation

Test Data. For testing the on-board module of SatZB model, the input data of a test case could be the location data generated by the submodule `LocUnit` and/or the commands generated by the submodule `TCC` according to Figure 8.6. However, in SatZB model, the generation of commands in the model of the traffic control centre is triggered by the received requests that were generated in accordance with the location data. Hence, the commands are seen as the responses to the reception of location data from the localisation unit. Assuming that the model of the traffic control centre is fully reliable (in fact, this is easy to be ensured in the test model, see Figure 8.8), then the input data of a test case for testing the on-board module takes only the location data into account. The location data represent the location of the train on the rail track, which is divided into block sections. The operational activities of a train control system depend on the current location of the train and the current state of the related block sections. Figure 8.9 shows a sketch of a rail track including three block sections S_a,S_b and S_c, and there exists a MA point in each block section represented by the sign of triangle.

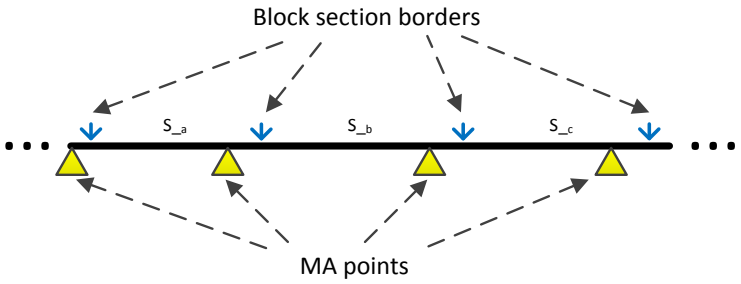


FIGURE 8.9: Sketch of a rail track

In this context, the black-box testing techniques *Equivalence Partitioning* (EP) and *Boundary Value Analysis* (BVA) [151] [152] are adopted to generate test data. Equivalence Partitioning is a testing technique that divides the input data of the SUT into partitions of data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. Equivalence Partitioning is based on the premise that the inputs and outputs of a SUT can be partitioned into classes that, according to the specification of the SUT, will be treated similarly by the SUT. Thus the result of testing a single value from an equivalence partition is considered representative of the complete partition and reducing the total number of test cases that must be developed. To make sure that the generated test cases are effective enough for the SUT, the testing technique Boundary Value Analysis is usually used combined with Equivalence Partitioning to generate test data. Boundary Value Analysis is a testing design technique in which tests are designed to include representatives of boundary values. Based

on the premise that developers are prone to making errors in their treatment of the boundaries of partitions, values on the edge of an equivalence partition or at the smallest value on either side of an edge should be considered. As a first model of SatZB, we assume that the MA point of each block section is identical to the border of each block section which is shown in Figure 8.10. There are five partitions ($p1, p2, p3, p4, p5$) and four boundaries ($b1, b2, b3, b4$), which means nine test data should be derived from these partitions and boundaries. However, for functional testing of the train control system model, the location data could be sorted into two categories: location data `Pt` (e.g., $b1, b2, b3, b4$) and `notPt` (e.g., $p1, p2, p3, p4, p5$), meaning the train is in the position of a MA point and not a MA point, respectively. Therefore, two test data (i.e., one `notPt` and one `Pt`) are sufficient to test the system model. Nevertheless, considering `OB_SN_Standby` as the initially activated scenario net of the on-board module, at least three location data are needed (e.g., in a sequence of `notPt, Pt, notPt`) to achieve the probability of activating all transitions in the test model.

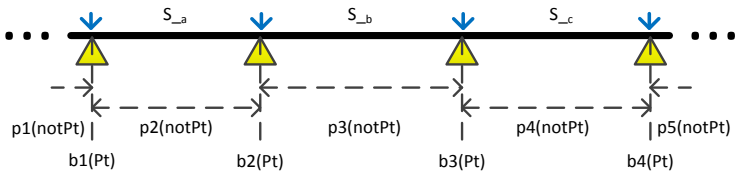


FIGURE 8.10: Partitions and boundaries of the rail track in Figure 8.9

Expected Outputs. To cover all scenario nets and possible scenario net transitions, the RT-based method introduced in chapter 5 could be used to generate the expected outputs according to the construction principles of the test model. As in this case, the state space is small and the reachability graph drawn by the CPN Tools [35] is relatively concise, we derive the expected outputs directly from the reachability graph without building another reachability tree. All possible paths of reachable states could be identified from the reachability graph. Each of such paths corresponds to a possible sequence diagrams or a possible combination of sequence diagrams. Accordingly, the expected output of a test case (i.e., tokens on specific places) can be extracted from a possible path. For the given example, the test data is obtained by initialising the place `GPS Data` in submodule `LocUnit` with a multiset [1] 1'(1,"notPt") ++1'(2,"Pt")++1'(3,"notPt"), and 24 paths are identified from the reachability graph shown in Figure 8.12. In other words, 24 expected outputs (e.g. tokens on places `Output Message` and `Input Message`) are derived for testing the on-board subsystem model. The state space

[1] A *multiset* m over a non-empty set S can be viewed as a function from S into the set of non-negative numbers $\mathbb{N}$. The function maps each element s into the number of appearances, $m(s)$, of the element s in the multiset m. The non-negative integer $m(s)$ is also called the *coefficient* of s in m[15].

report for the test model is shown in Figure 8.11, from which we can see that the reachability graph has 66 nodes and 84 arcs. Figure 8.13 shows the test case derived from the path shown in Figure 8.12, namely, node 1–node 2–node 3–node 4–node 8–node 12–node 16–node 21–node 35–node 40–node 45–node 52–node 65.

```
Statistics
----------------------------------

  State Space
     Nodes:  66
     Arcs:   84
     Secs:   0
     Status: Full

  Scc Graph
     Nodes:  66
     Arcs:   84
     Secs:   0
```

FIGURE 8.11: State space report: statistics

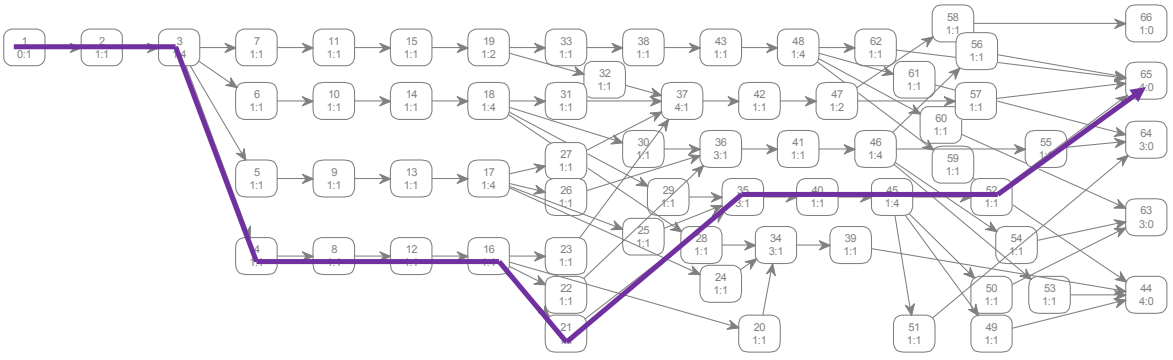


FIGURE 8.12: Reachability graph of the CPN model

8.4 Test Generation Based on SPENAT

Test generation by state space analysis has the potential of encountering the state explosion problem, especially for a complex system with concurrency. However, using SPENAT to generate test cases can void the state explosion problem but preserve the essence that the test cases cover the whole state space.

8.4.1 SPENAT

The SPENAT (Safe Place Transition Nets with Attributes [41], [116]) notation is built upon safe place transition nets (P/T nets) [153] and concepts of high-level Petri nets [15], [73], [154].

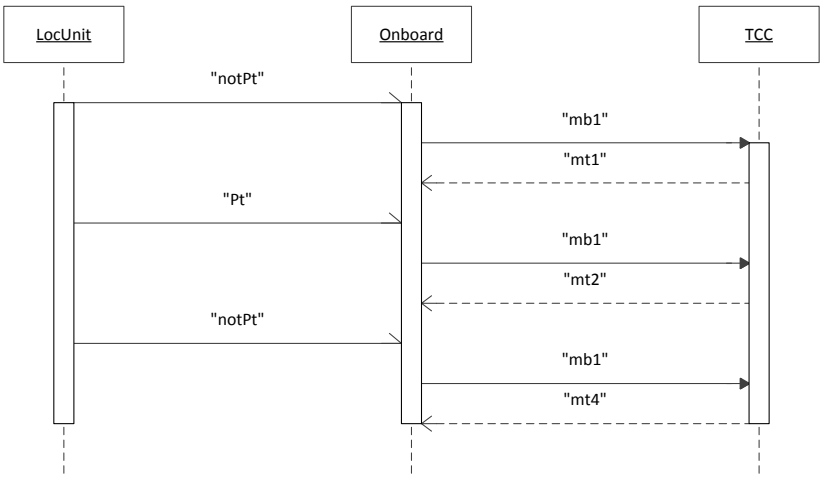


FIGURE 8.13: The test case extracted from the path shown in Figure 8.12

Places of SPENAT can only be marked with one (coloured) token at the same time and the arcs are not inscribed. Hierarchies are not allowed within SPENAT. Using SPENAT it is possible to use external and parameterised signal/events or ports as transition triggers (in contrast to STG [138], SIPN [153], IOPT [155]). Thanks to this feature it is much easier to model the required behaviour of an open and reactive system with a Petri net. Also, the mapping of existing models onto Petri nets is possible in an easy and intuitive way. For a SPENAT, it is able to use attributes with arbitrary data type for handling internal data states. With these features, one can model a system and/or component behaviour with a SPENAT like a well established input/output box. An example of the declaration of a SPENAT as a Petri net reacting on externally parameterised signals is presented in Figure 8.14. This SPENAT has two transitions where transition t_2 can only fire after transition t_1 has fired and the guard of t_2 depends implicitly on the value of the parameter x of the trigger event of t_1. If transition t_2 fires, then the parameter x of the external event $ev1(int\ x)$ must be 1. This value is a result of the guard of t_1 $(msg.x < 2)$, the effect of t_1 $(y = msg.x)$, and the guard of t_2 $(y > 0)$. The keyword msg is a reference to the respective trigger event of the transition. In this case the value 1 is the only valid value for parameter x of the trigger event $ev1(int\ x)$ that t_2 can fire.

8.4.2 Test Generation Based on Net Unfoldings

There are methods for verification and test generation based on SPENAT models. These methods use well-known Petri net techniques, especially the construction of the (complete)

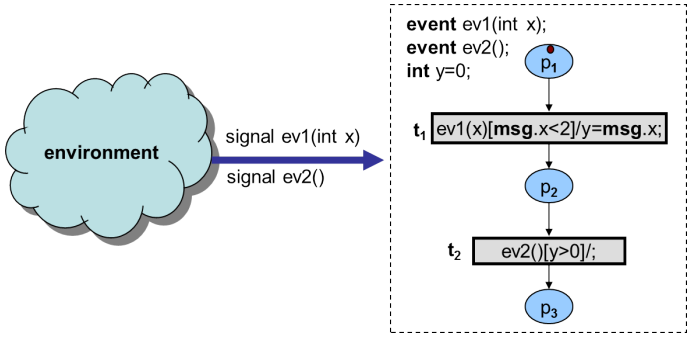


FIGURE 8.14: SPENAT with externally parameterized signals/events [41]

prefix of the Petri net unfolding [138]. The method of test generation based on SPENAT models uses the (complete) prefixes of the SPENAT unfoldings (unfolding-based test generation methods in general has been introduced in chapter 5). First a (maybe complete) prefix is constructed based on the specified test coverage criteria. It is not strictly necessary to construct the complete prefix of the unfolding of SPENAT. For instance, if it requires all transitions (or places) are covered by at least one test case, then the resulting prefix is in general much smaller than the complete prefix. After the prefix construction, the test cases are identified. The prefix of the unfolded SPENAT is an acyclic Petri net in which all possible processes of SPENAT are contained within the prefix. Each possible process can be identified by an event or rather by the *local configuration* of an event [119]. The local configurations of the events of the prefix that have no successor events represent maximum processes. Therefore, it is a good strategy to associate each identified maximum process with a test case. Here a process corresponds to a *linearisation* of a local configuration introduced in chapter 7.

The values of the external events as the stimuli of the test object are constrained by the inscription of the prefix events. When instantiating a process and assigning it to a test case, a value within the specified value range needs to be selected. This can be carried out in a random way but in general it is a widely accepted strategy to select a bound (upper and/or lower) within the specified value range. The identified test cases specify a (concurrent) message exchange between the test object and the tester or test system. This is an abstract sequence-based description of the stimuli and the expected responses of the test object. This abstract representation of the test cases must be transformed in an understandable and executable format for the test system. Furthermore, the realised level of abstraction during the modelling of the required test object behaviour must be respected in order to get automatically executable test specifications as a result of the test generation process. Data types, events, and/or signals, modelled within the profile model at an abstract level, have to be mapped to usable structures of the target test notation of the used test tool. Therefore, rules are necessary in order to automate this test formatting. For the formatting, in the standardised test notation TTCN-3 [156], suitable rules were developed and prototypical implemented.

8.4.3 Test Model

In [41] (see also [116]), it is shown that a UML state machine [157] can be mapped onto a SPENAT. Thus the use of a UML state machine as a specification model for generating test cases is also possible with SPENAT, whereby the UML state machine has to be mapped onto a semantic equivalent SPENAT. In this case, the state digram in Figure 3.14 is the specification model.

To develop a SPENAT model for test generation, a simple SPENAT modeller [41] can be used. With this tool, it is easy to declare SPENAT properties such as signals with parameters, ports, attributes, data types as text with a simple textual declaration notation. For the SPENAT model for testing the on-board module of SatZB model, 4 $enum$ types, 3 input signals with parameters, 2 ports and 1 attribute for handling the scenario nets of the on-board module are textually declared (see Figure 8.15). Parameter mt of the event $mov\_auth(MovAuth, mt)$ is a variable with the type of *enumerable*. Ports $pTCC$ and $pLoc$, representing the interfaces of the on-board module that interact with the model of the traffic control centre and the localisation unit respectively, are used to trigger the transitions in the SPENAT. Attribute $scenario$ indicates the currently activated scenario net of the on-board module. For the behaviour modelling, a simple graphical notation is used (see Figure 8.16). With this graphical notation, a normal Petri net is able to be modelled. Beside this, there are some simple extensions with respect to normal Petri net representations for convenience reasons. Transitions between two places with one pre- and one post- place are represented by a simple arc. *Junctions* for a better representations of possible branches are also used. They have the same syntax and semantics like junctions of a UML state machine.

8.4.4 Test Suite Generation

With the SPENAT modeller, the complete prefix of the SPENAT model of the on-board subsystem model can be computed. It has 64 conditions, 32 prefix events and 19 alternating processes. Therefore, 19 test cases were created for testing the on-board subsystem model. The calculation time for the complete prefix construction took 0.094s or 94 ms. In Figure 8.17, one generated test case is illustrated as a sequence diagram, in which one lifeline was inserted for each port.

declaration

```
1  type enum (Mt1, Mt2, Mt3,
   Mt4, Mt5) MovAuth;
2  type enum (Mb1, Mb2, Mb3,
   Mb4, Mb5) MovRequ;
3  type enum (Standby,
   Running,
   ConditionalRunning,
   BanOfEntry, EmergencyStop)
   ScenarioEnum;
4  type enum (Pt, notPt) →
   MaPt;
5
6  signal mov_auth(MovAuth
   mt);
7  signal mov_requ(MovRequ
   mb);
8  signal location(MaPt
   value);
9
10 port pTcc, pLoc;
11
12 attribute ScenarioEnum
   scenario=Standby;
```

FIGURE 8.15: The declarations of the SPENAT model

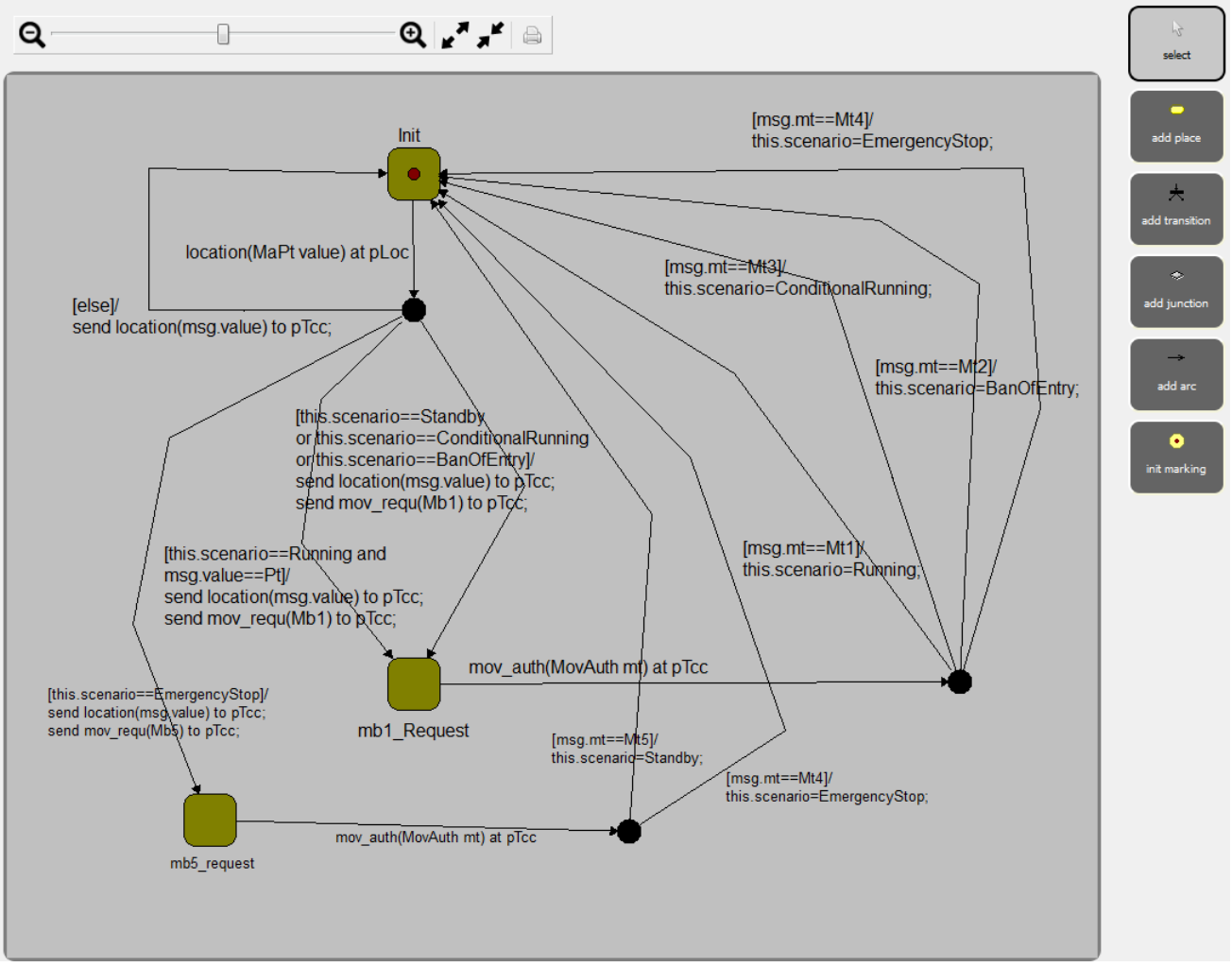


FIGURE 8.16: Screenshot of the SPENAT model for testing the on-board subsystem model

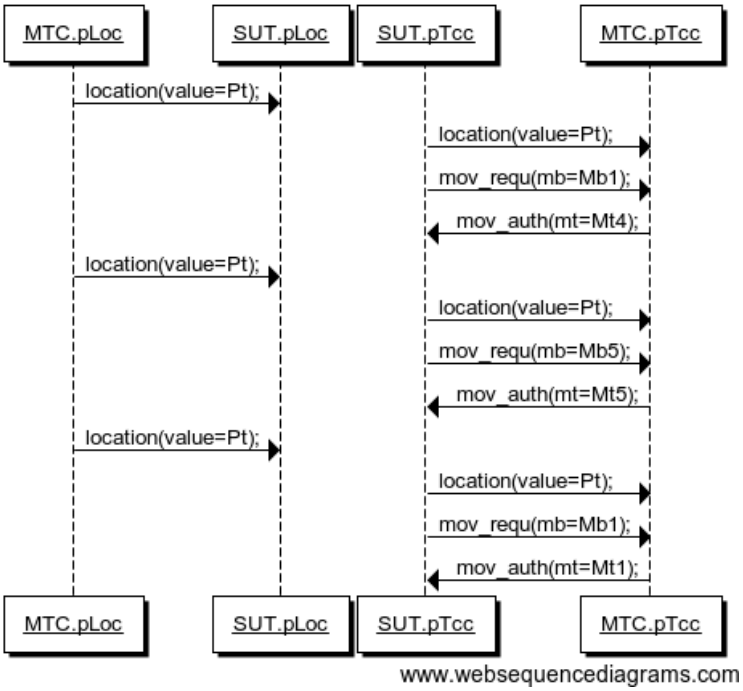


FIGURE 8.17: A generated test case with the on-board subsystem model as the SUT

8.4.5 Comparison of the Two Test Generation Techniques

Aiming at automatic test generation and test coverage control, two model-based approaches using CPNs and SPENAT respectively are proposed to test the on-board module of SatZB model. The test model developed with CPNs is a closed system including the environment that interacts with the on-board module, while the test model developed with SPENAT is an open system whose transitions are triggered by external events with attributes. As depicted in Figure 8.4, for the CPN-based approach, test cases are derived from the reachability graph that generated with the CPN Tools. As to the SPENAT-based approach, test cases are generated in a similar manner but with the use of the (complete) prefix of the SPENAT unfolding instead of the reachability graph. Table 8.1 presents some statistic data of the CPN model and the SPENAT model.

We can see that testing with CPNs may face the challenge of state space explosion problem. For example, when we shift the considering scenarios from the set STANDBY, RUNNING, and EMERGENCY_STOP to the set STANDBY, BAN_OF_ENTRY, CONDITIONAL_RUNNING, RUNNING, and EMERGENCY_STOP, the nodes of the reachability graph will grow from 39 to 66, and the paths of the reachability graph from 8 to 24. Testing with SPENAT could establish a model with less places and transitions. When the test object is simple enough, in this case few scenarios are under consideration, the number of test cases generated by both approaches are closed. However, if the test object is much more sophisticated, e.g., more scenarios are defined in this case, then huge number of test cases will generated by the CPN-based approach

TABLE 8.1: Statistic data of the CPN and SPENAT models

Test model		CPN model		SPENAT model	
Scenarios		STANDBY, RUNNING, EMERGENCY_STOP	STANDBY, BAN_OF_ENTRY, CONDITIONAL_RUNNING, RUNNING, EMERGENCY_STOP	STANDBY, RUNNING, EMERGENCY_STOP	STANDBY, BAN_OF_ENTRY, CONDITIONAL_RUNNING, RUNNING, EMERGENCY_STOP
Places		16	16	3	3
Transitions		14	18	11	13
Test cases		8	24	7	19
Reachability graph	Nodes	39	66	-	-
	Paths	8	24	-	-

while the number of test cases generated with SPENAT approach is in a reasonable scope. Nevertheless, the CPN-based approach has the advantage of saving the effort of constructing the test context.

In conclusion, both CPNs and SPENAT are high-level Petri nets, and their state spaces are represented by reachability graph and (complete) prefix of the Petri net unfolding, respectively. The reachability graph and (complete) prefix of the Petri net unfolding are used to generated test cases in CPN-based and SPENAT-based generation techniques, so these two techniques are essentially similar. However, if the number of states of the system is large and many of the states are concurrent, then the SPENAT-based approach is preferred.

8.5 Test Evaluation

As the generated test cases have the same abstract level as the test model (CPN model and SPENAT model), an adaptation of the test cases for test execution on the on-board subsystem model is necessary. For example, the message "mt1" in a test case has to be transformed into "{TRAINID=123,MSGID='mt1',TSTAMP=3,DATA='PERMIT RUNNING'}" for test execution. For test execution, the environment model (the CPN models of Figure 8.7 and Figure 8.8) could adopted as the environment of the on-board subsystem model. According to the observation of the test execution, tests with the both test suites (generated by CPN-based and SPENAT-based approaches, respectively) have passed. So no faults are detected by testing. Since the test cases cover all scenario nets and sequences of scenario net transitions, all requirements in the functional requirements specification are covered. Therefore, the following verification tasks identified in chapter 4 have been verified: (1)

the on-board module contains no algorithm errors; (2) *The on-board module can switch its activated scenario net to the net* `OB_SN_Emergency_Stop` *from any other nets (except for the nets* `OB_SN_Initialisation`, `OB_SN_Registration` *and* `OB_SN_Logout`*)*; (3) *The on-board module can switch its activated scenario net to the net* `OB_SN_Ban_Of_Entry` *from other nets (i.e.,* `OB_SN_Running` *and* `OB_SN_Conditional_Running`*)*; (4) *The on-board module can switch its activated scenario net to the net* `OB_SN_Conditional_Running` *from other nets (i.e.,* `OB_SN_Running`*)*; (5) *The on-board module has the possibility to activate every scenario net at least one time*; (6) *The on-board module switches its activated scenario net from one to another as Figure 3.7 depicted*; As a matter of fact, the task (6) has covered the tasks (2), (3), (4) and (5), so testing is one of the most efficient verification techniques.

8.6 Summary

This chapter first introduces the concept of testing and the test terminology, based on which model-based testing and its process are discussed, and then concentrates on two test generation techniques, i.e., CPN-based and SPENAT-based test generation techniques for testing the on-board module of SatZB model presented in chapter 3. The test cases generated by the CPN-based technique are derived from the reachability graph whereas the SPENAT-based technique generates test cases by the (complete) prefix of the SPENAT unfolding. Besides, the test model developed with CPNs is a closed system and the SPENAT model for testing is an open system whose transitions are possible to be triggered by the external and parameterised signal/events. A comparison of these two test generation techniques is carried out to show the advantages and disadvantages of both techniques, and a test evaluation is conducted in the end.

Chapter 9

Conclusions and Outlook

In this chapter, first a summary of the methods proposed in this thesis is provided. And then a verification table is suggested to guide the verification by Petri net techniques. After that approaches for the development and verification of the whole system (including the traffic control centre, the localisation unit and the mobile communication system) is discussed. Finally, the thesis conclude by outlining the future work.

9.1 Conclusions

Summary. System development is shifting from informal textual specifications and manual coding techniques to a model-based and tool-supported automated code generation process. In the model-based system development process, formal methods can be applied. This is, in particular, beneficial for safety-critical systems such as train control systems. Formal methods provide a means of developing a description of a system at some stage in the phases of requirements specification, design or coding. The resulting description takes a mathematical form and can be subjected to mathematical analysis to detect various classes of inconsistency or incorrectness. A formal method will generally offer a notation (formal language), a technique for deriving a description in that notation, and various forms of analysis for checking a description for different correctness properties. A modelling language is considered as formal if its syntax and semantics are expressed mathematically. It is easy to observe that the modelling language is the foundation of the formal methods. In this thesis, Petri nets (see chapter 2) are chosen as the modelling language for developing a satellite-based train control system, for their capability of describing concurrent and distributed systems, and their successful application on the railway domain. The hierarchical CPN model of the satellite-based train control system (see chapter 3) is established by following the BASYSNET method.

Before verifying the established system model of the train control system, the verification tasks have to be specified (see chapter 4). These tasks are identified based on the hazard analysis for the train control system and the system model, the identification of functions of the train control system for safety and normal operation as well as the allocation of these functions on the system model.

With the BASYSNET method for developing the train control system, it is possible to verify the system model by simulation, testing and formal analysis. This is called quality assurance in the approach. The quality assurance by Petri net analysing techniques (i.e., behavioural analysis, structural analysis, coverage-based test generation techniques) in generation is discussed in chapter 5.

In the conceptual model, a system has four representing properties: state, function, structure and behaviour. Each of these properties portrays a specific aspect of the system, so in order to fully verify a system (or system model), all the four properties of the system (model) should be verified. Different measures are required for the purpose of verifying different properties. For instance, the structural properties of the system model which are depend on the topological structure of the model can be analysed by structural analysis (see chapter 6); the system behaviour, in which states are involved, can be investigate by reachability analysis (see chapter 7); the functions of the system can be checked by functional testing (see chapter 8). Based on the characteristics of Petri nets and the modelling paradigms, specific techniques are proposed to verify the system model.

For the structural analysis of the system model, the concept of open nets is used to describe the scenario nets because of the modular structure of the system model. Open nets extend the normal Petri nets with open places (input/output boundary places) which serve as interfaces to their environments. The tokens on the open places represent messages that can be received but not yet received, or messages that are about to be sent out but not yet be sent out. Therefore, open places are given in terms of communication channels. Taking the environments of the open nets in into consideration, three application contexts are defined, i.e., normal-environment context, 1-configured-environment context and out-environment context. Based on the theory of structural properties of Petri nets introduced in chapter 5 and the verification tasks identified in chapter 4, the reproducibility of empty marking of open nets in different environmental contexts is discussed. In order to investigate the consistency of the scenario nets of the on-board subsystem model, a generic scenario net is constructed for applying the presented theory. For verifying the property of controllability of the scenario nets, the generic scenario net is adapted and the verification is performed by checking that whether the open nets in out-environment context will always end up with an empty marking.

For the behavioural analysis, unfolding-based techniques are employed to analyse the reachability property of the system model avoiding the state explosion problem. First the formalisation of unfolding hierarchical CP-nets based on the existing approach for non-hierarchical CP-nets is presented. And then the reachability property of the CPN model is analysed based on concept of linearisation and configuration. A configuration of an unfolding could have multiple linearisations, and each linearisation is a firing sequence of the occurrence net (from the default initial marking) containing each event from the configuration exactly once, and no further events. All linearisations lead to the same reachable marking. To describe a marking that is going to be checked in reachability analysis, the concept of partial marking is introduced. In particular, the on-the-fly verification approach is highlighted for checking a marking that is likely to be reachable. Otherwise, the complete prefix of the unfolding is required. The investigation of the reachability property of the on-board subsystem model is explored by giving examples verifying the verification tasks identified in chapter 4.

For the functional testing, two test generation techniques, CPN-based and SPENAT-based test generation techniques, are proposed. The test cases generated by the CPN-based technique are derived from the reachability graph whereas the SPENAT-based technique generates test cases with the (complete) prefix of the SPENAT unfolding. Besides, the test model developed with CPNs is a closed system and the SPENAT model for testing is an open system whose transitions are possible to be triggered by the external and parameterised signal/events. These techniques are exemplified by testing the on-board subsystem model with respect to the verification tasks identified in chapter 4.

In summary, all the verification tasks identified in chapter 4 are verified by either structural verification, reachability analysis or testing. According to the verification results, no errors were found in the on-board module of SatZB model.

Verification Table. Based on the introduction of S-invariants and T-invariants in chapter 2, behavioural and structural properties in chapter 5, and the structural analysis in chapter 6, a compact verification table, providing an overview of system verification by Petri net analysing techniques, can be given as in Table 9.1. In practice, system analysis can be performed by filling out this table. In this table, the second column (split by ||) shows the Petri nets under consideration and the Petri net properties in general. The transitions of a Petri net can be classified into three groups: input boundary transitions (indicated by the letter "I"), output boundary transitions (indicated by the letter "O") and normal (internal) transitions (indicated by the letter "N"). Similarly, the places of the net can be assigned into three category: input boundary places (indicated by the letter "I"), output boundary places (indicated by the letter "O") and normal (internal) places (indicated by the letter "N"). The first column of the table lists the T-invariants of the Petri nets. These T-invariants can be used to verify

the structural properties of consistency and controllability (see chapter 6) of the Petri nets. The third column presents the S-invariants of the Petri nets, which can be used to verify the structural property of conservativeness. The last column defines scenarios for behavioural analysis. The word "scenarios" in this table has a general meaning. It is not restricted to implying the scenarios of train control systems. A scenario is a process that has an initial state (indicated by "Init") and a end state (indicated by "End") if there is one. The expected states and unexpected states that might be appeared in a scenario are indicated by "Exp" and "Unexp", respectively. With these scenarios, the behavioural properties of boundedness, liveness, reversibility and reachability can be verified.

TABLE 9.1: Petri net verification table

Structural analysis			Petri nets		Structural analysis			Behavioural analysis											
T-invariants			Transition	Place	S-invariants			Scenarios											
								1				2				3			
3	2	1			1	2	3	Init	Exp	Unexp	End	Init	Exp	Unexp	End	Init	Exp	Unexp	End
			I↑-O↓																
				I↑-O↓															
			N↓-O↑																
				N↓-O↑															
			Consistency		-			-				-				-			
			Controllability		-			-				-				-			
-			Conservativeness					-				-				-			
-			Boundedness		-														
-			Liveness		-														
-			Reversibility		-														
-			Reachability		-														

Discussion. In this thesis, a verifiable design of a satellite-based train control system (SatZB) with Petri nets is proposed. Following the BASYSNET method, simulation, testing, formal analysis (including structural analysis and reachability analysis) are implemented to verify and validate the designed model by exploring corresponding methods. Although this thesis focus on the on-board module of the system model, the proposed verification methods are also valid to the other main module of the system model, the model of the traffic control centre. This is because it is also hierarchically constructed based on the exact scenarios that have been defined for constructing the on-board module (see chapter 3). Nevertheless, the structural verification for the model of the traffic control centre may be also interested in other structural properties such as boundedness, since the traffic control centre has to manage all the trains that have registered in it. Thus whether there are no overflows in the model of the traffic control centre is interested for the verification. The localisation unit and the communication system of SatZB system are developed by my colleagues who have rich knowledge

of GNSS-based localisation and mobile communication respectively, so the modelling and verification of these modules are carried out separately, e.g. in [158] and [159].

Recalling the introduction of the design and development of some existing train control systems in chapter 1, a comparison of the design and development processes of SatZB with other train control systems is shown in Table 9.2. It is easy to observe that the development process of SatZB is essentially similar to the development process of openETCS. Theses processes have the major advantages of formal verification and automatic code generation.

TABLE 9.2: Comparison of the design and development of different train control systems

	ETCS	openETCS	Satellite-based Train Control System		
			RZL	SATLOC	*SatZB*
Method	Object-oriented (Modularised and layered)	Open proof	UML-based	UML-based	*BASYSNET*
Means of description	Petri nets	SysML/ SCADE	UML	UML	*Petri nets*
Verification & validation	Test	Simulation, formal analysis, test	Test	Test	*Simulation, formal analysis, test*
Code generation	Manually	Automatically	Manually	Manually	*Automatically (out of the scope of this thesis)*

9.2 Outlook

For future work, the time constrains of the system could be taken into consideration. In other words, the time constrains could be added to the existing model, which results in a timed Petri net model. Given a timed Petri net model, its performances (e.g., time delay) are able to be analysed. Beside adding time constrains to the existing model, further refinement for the model should be done for the purpose of transforming the system model into programming codes directly. As a consequence, the identification of verification tasks could be further refined accordingly.

The discussion of the structural analysis in this thesis is based on the low-level Petri nets. The SatZB model established in chapter 3, however, is a high-level Petri net model. Therefore, a mapping from the high-level Petri nets to the low-level Petri nets is required before applying

the proposed methods. It needs to be aware that the mapped low-level Petri nets should preserve the structural properties of the corresponding high-level Petri nets.

For reachability analysis using net unfoldings, it is desirable to use computer-aided tools for automatically unfolding a (coloured) Petri net model and checking the reachability properties since unfolding the (coloured) Petri net model by hand is a time-consuming and error-prone process. Existing tools like Cunf [83] carries out unfolding-based verification of Petri nets extended with read arcs, also known as contextual nets (or c-nets) [160] which are low-level Petri nets. Moreover, the Cunf tool requires that the input c-nets is 1-safe. However, there are tools, e.g., CPN-AMI [80] and Snoopy [161], [162], that can unfold coloured Petri nets into low-level Petri nets (P/T nets).

Functional verification by testing with the approach presented in this thesis could be adapted to test the model of the traffic control centre and the whole system model. After adding time constrains to the system model, performance testing also could be one of the testing tasks. With the Petri net model of the system, test scenarios can be derived from the reachability graph of the system model. These test scenarios then applied to the real/realised system for system validation. From the perspective of testing, the design model can be taken as a test model for testing the realised system that might has been developed separately.

Apart from qualitative analysis, quantitative analysis of the risk and availability of the system can be performed with the PROFUND (PROcess FUNctional and Dependability) approach [163] by means of stochastic Petri nets.

Recently, a report on the Model Checking Contest at Petri Nets 2013 [164] was released. This report presents the results of the Model Checking Contest held at Petri Nets 2013 in Milano, Italy. The contest aimed at a fair and experimental evaluation of the performances of model checking techniques applied to Petri nets. Twelve tools were involved in several examinations (state space generation and evaluation of several types of formulae - reachability, LTL (Linear-time Temporal Logic) [165], [166], CTL (Computation Tree Logic) [167] for various classes of atomic propositions) run on a set of common models (Place/Transition and Symmetric Petri nets). The results presented on the report shows that no tool (among the twelve selected tools) could process the verified formulas of CTL and LTL for CPN models. However, the report suggests that a solution should be proposed to have high-level Petri nets. This should be possible when we first represent high-level Petri nets with PNML (Petri Net Markup Language) (e.g., CPN Tools 4.0 [35] supports for export to PNML), and then compile the PNML of the high-level Petri nets with certain tools such as `Neco` [168]. `Neco` is a suite of Unix tools to compile high-level Petri nets into libraries for explicit model-checking. These libraries can be used to build state spaces. In addition, a LTL model-check `Neco-spot` is

available. Although no tool provided can process the formulas of CTL and LTL for CPN models, the Design/CPN (has been replaced by CPN Tools [169]) is facilitated with the capability of analysing state space by means of a CTL-like temporal logic called ASK-CTL [170], [171], [172], which is an extension of CTL. It is not only possible to formulate queries about states, but also queries about state changes (e.g., the occurrence of certain transitions) [173]. Therefore, LTL, CTL or ASK-CTL model checking for the established CPN model of the satellite-based train control system could also be a future work.

Bibliography

[1] B. Vincze and G. Tarnai. Evolution of Train Control Systems. In *14th International Symposium EURNEX-ZEL*. Zilina and Slovak Republic, 2006.

[2] European Commission. 2006/860/EC: Commission Decision of 7 November 2006 concerning a technical specification for interoperability relating to the control-command and signalling subsystem of the trans-European high speed rail system and modifying Annex A to Decision 2006/679/EC concerning the technical specification for interoperability relating to the control-command and signalling subsystem of the trans-European conventional rail system (notified under document number C(2006) 5211) (Text with EEA relevance), 2006. URL http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32006D0860:EN:NOT. [checked on 2013-11-26].

[3] International union of railways. UIC Homepage: ETCS, 2013-03-04. URL http://www.uic.org/spip.php?rubrique869. [checked on 2013-11-26].

[4] H. Dong, B. Ning, B. Cai, and Z. Hou. Automatic Train Control System Development and Simulation for High-Speed Railways. *IEEE Circuits and Systems Magazine*, 10(2): 6–18, 2010. doi: 10.1109/MCAS.2010.936782.

[5] International union of railways. UIC Homepage: ERTMS Regional project, 2013-03-04. URL http://www.uic.org/spip.php?article420. [checked on 2013-11-26].

[6] Pierre Mertens, Jean-Pierre Franckart, and Antonin Starck. LOCOPROL: A low cost Train Location and Signalling system for "Low Density" Lines. *Retrieved(11 October 2008) from http://www. ertico. com/download/locoprol% 20doc uments/wcrr03v3. pdf*, 2003.

[7] B. Stadlmann, F. Kaiser, and S. Maihofer. Rechnergestütztes Zugleitsystem für die Pinzgauer Lokalbahn. *Signal+Draht*, (5):28–33, 2012.

[8] B. Stadlmann. Systematic UML-design used in the development of a basic train control system for regional branch lines. In *FORMS / FORMAT 2004*, pages 71–78. Beyrich DigitalService, Germany, 2004. ISBN 3-9803363-8-7.

[9] M. Meyer zu Hörste, I. Weber, I. Illgen, H. Schrom, and E. Schnieder. Distributed multi-train simulation with simulators for real components. In *Proceedings of Computers in Railways VII 2000- COMPRAIL 2000*, pages 1291–1300. WIT Press, Bologna and Italy, 2000. ISBN 1-85312-826-0.

[10] International union of railways. UIC Homepage: Project List, 2013. URL http://www.uic.org/apps/?module=uicprojects_home. [checked on 2013-11-26].

[11] M. Meyer zu Hörste, B. Ptok, E. Schnieder, and H. Schrom. A case study for the automated system development: the satellite-based train control system. In *Proceedings of IFAC Conference on Control Systems Design (CSD'2000)*, pages 329–334. Bratislava and Slovak Republic, 2000.

[12] L. Jansen, M. Meyer zu Hörste, and E. Schnieder. Technical issues in modelling the European train control system (ETCS) using coloured petri nets and the Design/CPN tools. In *Proceedings of the workshop on practical use of coloured Petri Nets and Design /CPN*, pages 103–115. Aarhus and Denmark, 1998.

[13] A. Janhsen, K. Lemmer, B. Ptok, and E. Schnieder. Formal specifications of the european train control system. In *Proceedings of 8th IFAC Symposium on Transportation Systems*, pages 1215–1220. Chania, 1997.

[14] M. Meyer zu Hörste. Modelling and Simulation of Train Control Systems using Petri Nets. In *FMRail Workshop*, volume 3. St. Pölten and Österreich, 1999.

[15] K. Jensen and L. M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer, 2009. ISBN 978-3-642-00283-0.

[16] K. R. Hase. "Open Proof" for Railway Safety Software – A Potential Way-Out of Vendor Lock-in Advancing to Standardization, Transparency, and Software Security. In *FORMS/FORMAT 2010*, pages 5–38. Springer, Braunschweig and Germany, 2011. ISBN 978-3-642-14260-4.

[17] openETCS. European Train Control System (ETCS): Open Proofs - Open Source, 2012. URL http://openetcs.org/. [checked on 2013-11-27].

[18] International union of railways. UIC Homepage: OpenETCS, 2013-03-04. URL http://www.uic.org/spip.php?article2998. [checked on 2013-11-27].

[19] European Railway Agency. SUBSET-026 v300, 2011-06-27. URL http://www.era.europa.eu/Document-Register/Pages/SUBSET-026v300.aspx. [checked on 2013-11-28].

[20] S. Baro and J. Welte. Requirements for openETCS: OETCS/WP2/D2.6-9 – 2.0.0, 2013. URL https://github.com/openETCS/requirements/commit/10842794bd9acc8e3a55ced46b2b5be8d97afbe0. [checked on 2013-11-28].

[21] CENELEC. EN 50128:2001. Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems, 2001.

[22] CENELEC. EN 50126-1:1999. Railway applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - Part 1: Basic requirements and generic process, 1999.

[23] ERTMS solutions: ERTMSFormalSpecs - Open Source. URL http://www.ertmssolutions.com/. [checked on 2012-12-02].

[24] L. Ferier, S. Pinte, and D. Blasband. ERTMS Formal Specs: a domain specific language to formalize ERTMS specifications for onboard unit development. 2011.

[25] ClearSy. ATELIER B, the industrial tool to efficiently deploy the B method, 2011. URL http://www.atelierb.eu/. [checked on 2013-11-19].

[26] Object Management Group. Documents Associated With SysML: Version 1.2, June 2010. URL http://www.omg.org/spec/SysML/1.2/. [checked on 2013-11-28].

[27] M. Jastram and M. Petit-Doche. Report on the Final Choice of the Primary Toolchain: Decision on the final choice for the means of description (O7.1.4), tools (O7.1.8) and tool platform (O7.1.11): OETCS/WP7/D7.1 – 02/00, October 2013. URL https://github.com/openETCS/toolchain/blob/master/Deliverables/D7.1.pdf. [checked on 2013-11-28].

[28] Object Management Group. Unified Modeling Language, 2013-09-13. URL http://www.uml.org/. [checked on 2013-11-19].

[29] SCSK Cooperation. VDMTools: VDMTools User Manual (VDM++), 2013. URL http://www.vdmtools.jp/uploads/manuals/usermanpp_a4E.pdf. [checked on 2013-12-16].

[30] SCSK Cooperation. VDMTools: VDMTools User Manual (VDM-SL), 2013. URL http://www.vdmtools.jp/uploads/manuals/usermansl_a4E.pdf. [checked on 2013-12-16].

[31] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, Cambridge and Massachusetts, 2008. ISBN 978-0-262-02649-9.

[32] E. Schnieder, G. Bikker, M. Chouikha, S. Einer, and M. Meyer zu Hörste. *BASYSNET-An integrated Approach for automated Control System Development*. Springer, 2003.

[33] C. Girault and R. Valk. *Petri Nets for Systems Engineering: A Guide to Modeling, Verfication, and Applications*. Springer, 2001.

[34] Eckehard Schnieder. *Methoden der Automatisierung: Beschreibungsmittel, Modellkonzepte und Werkzeuge für Automatisierungssysteme*. Studium Technik. Vieweg, 1999. ISBN 3-528-06566-4.

[35] Aarhus University. CPN Tools Homepage, 2000. URL http://cpntools.org/. [checked on 2013-11-19].

[36] J. Padberg, L. Jansen, R. HECKEL, and H. EHRIG. Interoperability in train control systems: specification of scenarios using open nets. In *Proceedings of the 3rd Biennial World Conference on Integrated Design and Process Technology (IDPT)*, volume 98, pages 17–28. Berlin and Germany, 1998.

[37] L. Jansen, E. Schnieder, J. Padberg, H. EHRIG, and R. HECKEL. Cooperability in train control systems: Specification of scenarios using open nets. *Journal of Integrated Design and Process Science*, 5(1):3–21, 2001.

[38] K. Lautenbach. Reproducibility of the Empty Marking. *Application and Theory of Petri Nets 2002*, pages 281–300, 2002.

[39] V. Januzaj. CPNunf: A tool for McMillan's Unfolding of Coloured Petri Nets. In *Proceedings of 8th Workshop on Practical use of Coloured Petri Nets and the CPN Tools*, pages 147–166. 2007.

[40] J. Esparza and K. Heljanko. A New Unfolding Approach to LTL Model Checking. *Automata, Languages and Programming*, pages 475–486, 2000.

[41] J. Krause. *Testfallgenerierung aus modellbasierten Systemspezifikationen auf der Basis von Petrinetzentfaltungen*. PhD thesis, Universität Magdeburg, 2012.

[42] Institute for Quality, Safety and Transportation. Pi-Tool. URL http://www.iqst.de/?page_id=24. [checked on 2013-11-19].

[43] K. Lautenbach, J. R. Müller, and S. Philippi. Modellierung, Simulation und Analyse mit dem Petri-Netz-Tool POSEIDON. In *Promise*, pages 163–174. 2002.

[44] S. Tjell. *Formal Requirements Modeling for Reactive Systems with Coloured Petri Nets*. PhD thesis, University of Aarhus, Denmark, 2009.

[45] R. Zurawski and M. Zhou. Petri Nets and Industrial Applications: A Tutorial. *IEEE Transactions on Industrial Electronics*, 41(6):567–583, 1994.

[46] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77 (4):541–580, 1989.

[47] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers*, 100(9):913–917, 1982.

[48] B. Berthomieu and M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

[49] K. Jensen, L. M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007. doi: 10.1007/s10009-007-0038-x.

[50] K. Garg. An Approach to Performance Specification of Communication Protocols Using Timed Petri Nets. *IEEE Transactions on Software Engineering*, SE-11(10):1216–1225, 1985.

[51] S. S. Yan and M. U. Caglayan. Distributed Software System Design Representation Using Modified Petri Nets. *IEEE Transactions on Software Engineering*, SE-9(6):733–745, 1983.

[52] S. Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Transactions on Software Engineering*, 32(7): 486–502, 2006.

[53] Y. E. Papelis and T. L. Casavant. Specification and Analysis of Parallel/Distributed Software and Systems by Petri Nets with Transition Enabling Functions. *IEEE Transactions on Software Engineering*, 18(3):252–261, 1992.

[54] P. Azema, G. Juanole, E. Sanchis, and M. Montbernard. Specification and verification of distributed systems using prolog interpreted petri nets. In *Proceedings of the 7th international conference on Software engineering*, pages 510–518. 1984.

[55] C. V. Ramamoorthy and G. S. Ho. Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. *IEEE Transactions on Software Engineering*, SE-6(5):440–449, 1980.

[56] V. E. Kotov. An algebra for parallelism based on petri nets. In *Mathematical Foundations of Computer Science 1978*, pages 39–55. 1978.

[57] T. Kozlowski, E. L. Dagless, J. M. Saul, and J. Szajna. Parallel controller synthesis using Petri nets. In *IEE Proceedings- Computers and Digital Techniques*, volume 142, pages 263–271. 1995.

[58] R. M. Keller. Formal Verification of Parallel Programs. *Communications of the ACM*, 19 (7):371–384, 1976.

[59] G. Balbo, G. Chiola, S. C. Bruell, and P. Chen. An Example of Modeling and Evaluation of a Concurrent Program Using Colored Stochastic Petri Nets: Lamport's Fast Mutual Exclusion Algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):221–240, 1992.

[60] K. Lautenbach and H. Ridder. The Linear Algebra of Deadlock Avoidance– A Petri Net Approach. *Inst. Softw. Technol., Univ. Koblenz-Landau, Koblenz, Germany, Tech. Rep*, (25-1996), 1996.

[61] K. Heljanko. Bounded Reachability Checking with Process Semantics. *CONCUR 2001—Concurrency Theory*, pages 218–232, 2001.

[62] T. Murata, B. Shenker, and S. M. Shatz. Detection of Ada Static Deadlocks Using Petri Net Invariants. *IEEE Transactions on Software Engineering*, 15(3):314–326, 1989.

[63] W. M. Zuberek. Timed Petri nets and preliminary performance evaluation. In *Proceedings of the 7th annual symposium on Computer Architecture*, pages 88–96. 1980.

[64] C. Ramchandani. Analysis of asynchronous concurrent systems by timed Petri nets. 1974.

[65] J. Sifakis. Use of Petri nets for performance evaluation. *Acta Cybern*, 4:185–202.

[66] R. Zijal, G. Ciardo, and G. Hommel. *Discrete Deterministic and Stochastic Petri Nets*. DTIC Document. 1996.

[67] M. K. Molloy. Discrete time stochastic Petri nets. *IEEE Transactions on Software Engineering*, (4):417–423, 1985.

[68] M. A. Marsan and G. Conte. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems (TOCS)*, 2(2):93–122, 1984.

[69] M. A. Holliday and M. K. Vernon. A generalized timed Petri net model for performance analysis. *IEEE Transactions on Software Engineering*, (12):1297–1310, 1987.

[70] M. A. Marsan and G. Chiola. On petri nets with deterministic and exponentially distributed firing times. In *Advances in Petri Nets 1987*, pages 132–145. Springer, 1987.

[71] R. Zijal. Discrete Time Deterministic and Stochastic Petri Nets. In *Quality of Communication-Based Systems*, pages 123–136. Springer, 1995.

[72] E. Best, H. Fleischhack, W. Fraczak, R. P. Hopkins, H. Klaudel, and E. Pelz. *An M-net Semantics of B(PN)2*. Structures in Concurrency Theory. Springer, 1995.

[73] E. Best, H. Fleischhack, W. Fraczak, R. Hopkins, H. Klaudel, and E. Pelz. A Class of Composable High Level Petri Nets. In *ATPN' 95*. Springer, 1995.

[74] V. Khomenko and M. Koutny. Branching Process of High-Level Petri Nets. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 458–472, 2003.

[75] The Fellowship of SML/NJ. Standard ML of New Jersey. URL http://www.smlnj.org/. [checked on 2013-11-19].

[76] TGI group. Petri Nets World. URL http://www.informatik.uni-hamburg.de/TGI/PetriNets/. [checked on 2013-11-19].

[77] Imperial College London. PIPE2: Platform Independent Petri net Editor 2. URL http://pipe2.sourceforge.net/. [checked on 2013-11-19].

[78] Tchnische Universität Ilmenau. TimeNET. URL http://www.tu-ilmenau.de/TimeNET. [checked on 2013-11-19].

[79] University of Geneva. AlPiNA: an Algebraic Petri Net Analyzer. URL http://alpina.unige.ch/. [checked on 2013-11-19].

[80] University Pierre & Marie Curie. CPN-AMI, 2010-10-19. URL http://www.lip6.fr/cpn-ami. [checked on 2013-11-19].

[81] Humboldt-Universität zu Berlin. INA: Integrated Net Analyzer, 2003-07-31. URL http://www.informatik.hu-berlin.de/~starke/ina.html. [checked on 2013-11-19].

[82] Università di Torino. GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets: GreatSPN, 2008-09-25. URL http://www.di.unito.it/~greatspn/. [checked on 2013-11-19].

[83] César Rodríguez. Cunf: A toolset for unfolding-based verification of Petri nets with read arcs. URL http://code.google.com/p/cunf/. [checked on 2013-11-19].

[84] FernUniversität in Hagen. VIP Tool. URL http://www.fernuni-hagen.de/sttp/forschung/vip_tool.shtml. [checked on 2014-03-18].

[85] F. Mejia. Formalizing existing safety-critical software. In *FMERail Workshop*, volume 2. 1998.

[86] K. Rástočný, A. Janota, and J. Zahradník. The Use of UML for Development of a Railway Interlocking System. In *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*, pages 174–198. Springer, 2004.

[87] M. Fukuda, Y. Hirao, and T. Ogino. VDM specification of an interlocking system and a simulator for its validation. In *Proceedings of 9th IFAC Symposium on Control in Transportation Systems 2000*, pages 187–192. Braunschweig and Germany, 2000.

[88] M. Meyer zu Hörste. *Methodische Analyse und generische Modellierung von Eisenbahnleit- und -sicherungssystemen*. PhD thesis, Technische Universität Braunschweig, Germany, 2003.

[89] M. Fessler and J. Schütte. DIE METHODE B Sichere Software für Bahnanwendungen. In *Forms'99 – Formale Techniken für die Eisenbahnsicherung*, pages 273–281. VDI Verlag GmbH, Düsseldorf and Germany, 2000. ISBN 3-18-343612-4.

[90] ClearSy. Atelier B 4 User manual: Version 4.0. URL www.atelierb.eu/manuels/manuel-utilisateur-atelier-b-4.0-en.pdf. [checked on 2013-12-16].

[91] D. T. Ross. Structured analysis (SA): A language for communicating ideas. *IEEE Transactions on Software Engineering*, (1):16–34.

[92] G. Barbu. SADT for elaboration and assessment of functional specifications of GNSS supported operation on low traffic density lines. In *Proceedings of Symposium FORMS/-FORMAT 2008*. L'Harmattan, Budapest and Hungary, 2008. ISBN 978-963-236-138-3.

[93] A. Janota and J. Zahradník. UML-An object oriented approach to formal specification of safety relevant systems. In *4th international scientific conference ELEKTRO 2001*. Žilina and Slovak Republic, 2001.

[94] IBM Cooperation. Systems Engineering Tutorial for Rational Rhapsody, 1997. URL http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/topic/com.ibm.help.download.rhapsody.doc/pdf75/tutorial_Systems_Eng.pdf. [checked on 2013-12-16].

[95] K. Jensen, S. Christensen, and L. M. Kristensen. CPN Tools State Space Manual, 2006. URL http://cpntools.org/_media/documentation/manual.pdf. [checked on 2013-12-16].

[96] K. H. Mortensen. *Automatic code generation method based on coloured petri net models applied on an access control system*. Application and Theory of Petri Nets 2000. Springer, 2000.

[97] K. M. Hansen. Validation of a Railway Interlocking Model. In *FME'94: Industrial Benefit of Formal Methods*, volume 873, pages 582–601. 1994.

[98] L. Schnieder and S. Detering. Systems- Theoretic Foundation for Advanced Driver Assistance Systems. In *2010 4th Annual IEEE Systems Conference*, pages 365–370. 2010.

[99] J. Drewes, J. May, E. Schnieder, and N. König. Structured Approach of A Generic (Signalling) Hazard List for Railway (Interlocking) Systems. In *Proceedings of the 5th European Congress and Exhibition on Intelligent Transport Systems and Services*. Hannover and Germany, 2005.

[100] M. Pope, J. Drewes, and J. May. Generic Hazard List for Railway Systems. In *7th World Congress on Railway Research- WCRR 2006*. Montréal and Canada, 2006.

[101] N. König, J. Drewes, and J. May. Structured approach for a generic signalling hazard list for railway (interlocking) systems using formal methods. In *FORMS / FORMAT 2004*, pages 61–67. Beyrich DigitalService, Germany, 2004. ISBN 3-9803363-8-7.

[102] J. Drewes. *Verkehrssicherheit im systemischen Kontext*. PhD thesis, Technische Universität Braunschweig, Braunschweig and Germany, 2009.

[103] E. Schnieder T. Ständer L. Schnieder. Railway Safety and Security-Two Sides of the Same Coin?! In *International Railway Safety Conference 2009*. Sweden, 2009.

[104] J. R. Müller. Die formalisierte Terminologie der Verlässlichkeit technischer Systeme: Habilitationsschrift, 2012.

[105] H. Zhu and X. He. A methodology of testing high-level Petri nets. *Information and Software Technology*, 44(8):473–489, 2002. ISSN 09505849. doi: 10.1016/S0950-5849(02)00048-4.

[106] H. Zhu and X. He. *A theory of behaviour observation in software testing*. Technical Report CMS-TR-99-05, School of Computing and Mathematical Sciences, Oxford Brookes University, 1999.

[107] H. J. Genrich. *Predicate/transition nets*. Petri Nets: Central Models and Their Properties. Springer, 1987.

[108] H. J. Genrich and K. Lautenbach. *The analysis of distributed systems by means of predicate/transition-nets*. Semantics of Concurrent Computation. Springer, 1979.

[109] N. Adjir, P. De Saqui-Sannes, and K. M. Rahmouni. *Testing real-time systems using TINA*. Testing of Software and Communication Systems. Springer, 2009.

[110] R. Lill and F. Saglietti. Test Coverage Criteria for Autonomous Mobile Systems based on Coloured Petri Nets. In *9th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2012)*, pages 155–162. Braunschweig and Germany, 2012.

[111] R. Lill and F. Saglietti. Model-based testing of autonomous systems based on Coloured Petri Nets. In *ARCS Workshops (ARCS), 2012*, pages 1–5. 2012.

[112] D. Xu. *A tool for automated test code generation from high-level Petri nets*. Applications and Theory of Petri Nets. Springer, 2011.

[113] J. Desel, A. Oberweis, T. Zimmer, and G. Zimmermann. Validation of Information System Models: Petri Nets and Test Case Generation. In *1997 IEEE International Conference on Computational Cybernetics and Simulation*, volume 4 of *Systems, Man, and Cybernetics, 1997*, pages 3401–3406. 1997.

[114] H. Watanabe and T. Kudoh. Test Suite Generation Methods for Concurrent Systems based on Coloured Petri Nets. In *Software Engineering Conference, 1995. Proceedings., 1995 Asia Pacific*, pages 242–251. 1995.

[115] K. Jensen. *Coloured Petri nets*, volume 254 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1987. ISBN 978-3-540-17905-4.

[116] J. Krause, E. Hintze, S. Magnus, and C. Diedrich. Model Based Specification, Verification, and Test Generation for a Safety Fieldbus Profile. In *Computer Safety, Reliability, and Security*, volume 7612 of *Lecture Notes in Computer Science*, pages 87–98. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33677-5.

[117] K. L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Computer Aided Verification*, pages 164–177. 1993.

[118] K.L. McMillan. A Technique of State Space Search Based on Unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[119] J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 87–106, 1996.

[120] J. L. Peterson. Petri Nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977.

[121] F.-Y Wang, Y. Gao, and M.-C Zhou. A Modified Reachability Tree Approach to Analysis of Unbounded Petri Nets. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 34(1):303–308, 2004. ISSN 1083-4419. doi: 10.1109/TSMCB.2003.811516.

[122] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.

[123] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. Reachability trees for high-level petri nets. *Theoretical Computer Science*, 45:261–292, 1986. doi: 10.1016/0304-3975(86)90046-0.

[124] J. Esparza and K. Heljanko. Implementing LTL Model Checking with Net Unfoldings. *Model Checking Software*, pages 37–56, 2001.

[125] B. Bonet, P. Haslum, S. Hickmott, and S. Thiebaux. Directed Unfolding of Petri Nets. *Transactions on Petri Nets and Other Models of Concurrency I*, pages 172–198, 2008.

[126] R. HECKEL. *Open Petri Nets as Semantic Model for Workflow Integration*. Petri Net Technology for Communication-Based Systems. Springer, 2003.

[127] P. BALDAN, A. CORRADINI, H. EHRIG, and R. HECKEL. Compositional semantics for open Petri nets based on deterministic processes. *Mathematical Structures in Computer Science*, 15(1):1–35, 2005. doi: 10.1017/S0960129504004311.

[128] P. BALDAN, A. CORRADINI, H. EHRIG, R. HECKEL, and B. König. *Bisimilarity and Behaviour-Preserving Reconfigurations of Open Petri Nets*. Algebra and Coalgebra in Computer Science. Springer, 2007.

[129] E. Best and H. A. Schmid. System of open paths in petri nets. In *Mathematical Foundations of Computer Science 1975 4th Symposium*. Mariánské Lázně, 1975. ISBN 978-3-540-37585-2.

[130] J. Esparza and K. Heljanko. *Unfoldings: A Partial-Order Approach to Model Checking*. Springer, 2008. ISBN 978-3-540-77425-9.

[131] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structure and domains, Part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[132] J. Esparza and C. Schröter. Unfolding Based Algorithms for the Reachability Problem. *Fundamenta Informaticae*, 47(3-4):231–245, 2001.

[133] C. Schröter and V. Khomenko. Parallel LTL-X Model Checking of High-Level Petri Nets Based on Unfoldings. In *Computer Aided Verification 2004*, pages 374–377. 2004.

[134] M. Notomi and T. Murata. Hierarchical Reachability Graph of Bounded Petri nets for Concurrent-Software Analysis. *IEEE Transactions on Software Engineering*, 20(5):325–336, 1994.

[135] S. Christensen and L. Petrucci. Modular State Space Analysis of Coloured Petri Nets. *Application and Theory of Petri Nets 1995*, pages 201–217, 1995.

[136] S. Melzer. *Verifikation verteilter Systeme mittles linearer - und Constraint-Programmierung*. PhD thesis, Technische Universitäte München, Germany, 1998.

[137] Keijo Heljanko. Using Logic Programs with Stable Model Semantics to Solve Deadlock and Rachability Problems for 1-safe Petri Nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.

[138] V. Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, University of Newcastle upon Tyne, UK, 2003.

[139] S. L. Hickmott. *Directed Unfolding: Reachability Analysis of Concurrent Systems & Applications to Automated Planning*. PhD thesis, University of Adelaide, Australia, 2008.

[140] A. Pretschner and J. Philipps. 10 Methodological Issues in Model-Based Testing. In *Model-based testing of reactive systems*, pages 11–18. Springer, 2005.

[141] Marc Horstmann. *Verflechtung von Test und Entwurf für eine verlässliche Entwicklung eingebetteter Systeme im Automobilbereich*. PhD thesis, Technische Universität Braunschweig, Germany, 2005.

[142] International Software Testing Qualifications Board (ISTQB). Standard glossary of terms used in Software Testing, October 19, 2012.

[143] M. Horstmann, E. Schnieder, P. Mäder, S. Nienaber, and H-M. Schulz. A framework for interlacing Test and/with Design. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pages 17–32. Edinburgh and Scotland, 2004.

[144] A. Pretschner and M. Leucker. 20 Model-Based Testing – A Glossary. In *Model-based testing of reactive systems*, pages 607–609. Springer, 2005.

[145] Software Program Managers Network. Little Book of Testing, Vol. I – Overview and Best Practices, June 1998. URL www.spmn.com. [checked on 2013-11-19].

[146] M. Meyer zu Horste, E. Schnieder, and H-M. Schulz. Test case generation from formal specifications on the example of train control systems. In *Proceedings of Computers in Railways VII 2000- COMPRAIL 2000*, pages 117–126. WIT Press, Bologna and Italy, 2000. ISBN 1-85312-826-0.

[147] M. Utting, A. Pretschner, and B. Legeard. *A taxonomy of model-based testing*. Working Paper Series. The University of Waikato, Hamilton and New Zealand, 2006.

[148] D. Hoffman. A taxonomy for test oracles. In *Quality Week*, volume 98, pages 52–60. 1998.

[149] Cem Kaner. A Course in Black Box Software Testing: Examples of Test Oracles, 2004. URL http://www.testingeducation.org/k04/OracleExamples.htm. [checked on 2013-11-19].

[150] IEEE Computer Society. IEEE Standard for Software and System Test Documentation, 18 July 2008.

[151] British Computer Society Specialist Interest Group in Software Testing (BCS SIGIST). Standard for Software Component Testing, 2001.

[152] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. Wiley, 2011.

[153] G. Frey. *Design and formal analysis of Petri net based logic control algorithms*. PhD thesis, Shaker Verlag, 2002.

[154] ISO/IEC 15909-1:2004. ISO/IEC 15909-1:2004 Systems and software engineering – High-level Petri nets – Part 1: Concepts, definitions and graphical notation, 2004.

[155] L. Gomes, J. P. Barros, A. Costa, and R. Nunes. The Input-Output Place-Transition Petri Net Class and Associated Tools. In *5th IEEE International Conference on Industrial Informatics*, volume 1, pages 509–514. 2007.

[156] European Telecommunications Standards Institute. Testing and Test Control Notation v.3 - TTCN-3, 2009. URL http://www.ttcn3.org/. [checked on 2013-10-08].

[157] Object Management Group. Documents associated with Unified Modeling Language (UML), v2.4: Superstructure Specification, 2011. URL http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF. [checked on 2013-12-16].

[158] D. Lu, H. Manz, J. R. Müller, and E. Schnieder. Integrity Verification for Satellite based Train Localisation Unit Structure by Means of Petri net. In *European Navigation Conference 2011 (ENC 2011)*. 2011.

[159] L. Chen, T. Tang, X. Zhao, and E. Schnieder. Verification of the safety communication protocol in train control system using colored Petri net. *Reliability Engineering & System Safety*, 100:8–18, 2012. ISSN 09518320. doi: 10.1016/j.ress.2011.12.010.

[160] P. BALDAN, A. Bruni, A. CORRADINI, B. König, C. Rodríguez, and S. Schwoon. Efficient unfolding of contextual Petri nets. *Theoretical Computer Science*, 449:2–22, 2012. doi: 10.1016/j.tcs.2012.04.046.

[161] Brandenburg University of Technology Cottbus. Snoopy. URL http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy. [checked on 2014-03-26].

[162] M. Heiner, M. Herajy, F. Liu, C. Rohr, and M. Schwarick. *Snoopy – A Unifying Petri Net Tool*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012.

[163] R. Slovák, J. May, and E. Schnieder. PROFUND modelling for holistic risk and availability analysis by means of stochastic Petri nets applied to a level crossing control system. In *Formal Methods for Railway Operation and Control Systems*, pages 221–232. Budapest and Hungary, 2003.

[164] F. Kordon, A. Linard, M. Beccuti, D. Buchs, Ł. Fronc, L. M. Hillah, F. Hulin-Hubard, F. Legond-Aubry, N. Lohmann, A. Marechal, E. Paviot-Adet, F. Pommereau, C. Rodríguez, C. Rohr, Y. Thierry-Mieg, H. Wimmel, and K. Wolf. Model checking contest at Petri nets contest: Report on the 2013 edition, 2013. URL http://arxiv.org/abs/1309.2485v1. [checked on 2013-11-19].

[165] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, 1977*, pages 46–57. 1977. ISBN 0272-5428.

[166] M. Y. Vardi. *An Automata-Theoretic Approach to Linear Temporal Logic*. Logics for concurrency. Springer, 1996.

[167] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[168] thelvyn.32. neco-net-compiler: Neco is a Petri net compiler based on SNAKES toolkit. URL http://code.google.com/p/neco-net-compiler/. [checked on 2013-11-19].

[169] University of Aarhus. Design/CPN: Computer Tool for Cloured Petri Nets, 2006-01-17. URL http://www.daimi.au.dk/designCPN/. [checked on 2013-11-20].

[170] A. Cheng, S. Christensen, and K. H. Mortensen. Model Checking Coloured Petri Nets Exploiting Strongly Connected Components. *technical report, University of Aarhus, Denmark*, 1996.

[171] A. Cheng, S. Christensen, and K. H. Mortensen. Model Checking Coloured Petri Nets Exploiting Strongly Connected Components. *DAIMI Report Series*, 26(519), 1997.

[172] S. Christensen and K. H. Mortensen. Design/CPN ASK-CTL Manual (Version 0.9), 1996. URL http://cpntools.org/_media/documentation/askctlmanual.pdf. [checked on 2013-12-16].

[173] Aarhus University. CPN Tools Homepage: Temporal logic for state space, 2000. URL http://cpntools.org/documentation/tasks/verification/temporal_logic_for_state. [checked on 2013-11-20].